



Universidade da Madeira

*Modelação de Diálogos com o Midiki:
um Gestor de Diálogo do tipo
Information State Update*

Lúcio M. M. Quintal

(Licenciado)

Tese submetida à Universidade da Madeira
para a obtenção do grau de Mestre
em Engenharia Informática

Funchal – Portugal

Novembro de 2008

Orientador:

Professor Doutor Paulo N.M. Sampaio

Professor Auxiliar do Departamento de Matemática e Engenharias,
Universidade da Madeira.

Resumo

Nesta dissertação investigamos a problemática da criação de sistemas e interfaces que permitam a interacção entre pessoas e máquinas através de linguagem natural (LN), recorrendo a Gestores de Diálogo (GD). Esse tipo de interacção concretiza-se através do estabelecimento de diálogos entre uma pessoa (cliente ou utilizador de um serviço) e a máquina, por exemplo, e em particular, através da fala.

Quando disponibilizado da forma tradicional, o acesso ao serviço exige um intermediário Humano ou a adaptação da Pessoa a interfaces menos naturais, tais como linhas de comandos num computador, digitadas através de teclado ou o recurso (usual) a janelas, cliques de rato e preenchimento de formulários. Os sistemas que possibilitam a intermediação com esses serviços através de LN chamam-se Sistemas de Diálogo (SD), no núcleo dos quais se encontram os chamados Gestores de Diálogo. A implementação de SDs robustos ainda constitui um desafio, dada a complexidade, problemas e dificuldades que apresenta. Um SD, e em particular um GD, tem de ser configurado para levar a cabo um diálogo em linguagem natural com um Humano, por mais restrito ou mais genérico que seja o domínio (ou tarefa) considerado. Infelizmente, existem poucas metodologias e ferramentas de autoria que possibilitem a modelação fácil e intuitiva de tais diálogos (sobre os GDs). Nesta dissertação apresentamos uma metodologia [Quintal & Sampaio, 2007] e uma ferramenta para a autoria de diálogos com base no Gestor de Diálogo MIDIKI [Burke, 2005b]. A ferramenta de autoria automatiza as partes mais importantes da *geração* de código com vista à execução de um diálogo nesse GD.

Palavras chave

Interacção Pessoa Máquina

Sistema de Diálogo

Gestão de Diálogo

Gestor de Diálogo

Interacção Natural

Interacção Multimodal

Information State Update

Metodologia de Modelação de Diálogos

Ferramenta de Modelação de Diálogos

Ferramenta de Autoria de Diálogos

Gestor de Diálogo Midiki

Abstract

In this thesis we investigate the issues related to the building of systems and interfaces that enable interaction between a person and a machine using natural language, through the use of Dialogue Managers (DM). This type of interaction is carried out through the dialogues between the person (client or user of a service) and the machine, for example, and particularly, through speech.

When provided in the traditional mode, access to the service requires a human mediator or the adaptation of the Person to less natural interfaces, such as command lines on a computer, entered/typed by using a keyboard or the common use of windows, mouse clicks, and filling of forms. Systems that enable this intermediation are called Dialogue Systems (DS), at the core of which we find the so-called Dialogue Managers. The implementation of a robust DS faces some challenges, given the complexity, problems and difficulties it presents. A DS, and in particular a DM, must be configured to carry out a dialogue in natural language with a Human, no matter how restricted or how generic is the domain (or task) being addressed. Unfortunately, there are few methodologies and tools that allow authors to carry out an intuitive and easy modeling of such dialogues (on the DMs). This thesis presents a methodology [Quintal & Sampaio, 2007] and a tool for the authoring of dialogues for the MIDIKI Dialogue Manager [Burke, 2005b]. The tool automates the most relevant parts in generating the code for carrying out a dialogue in that DM.

Keywords

Human Computer Interaction

Human Machine Interaction

Dialogue System

Dialogue Management

Dialogue Manager

Natural Interaction

Multimodal Interaction

Information State Update

Dialogue Modeling Methodology

Dialogue Modeling Tool

Dialogue Authoring Tool

Midiki Dialogue Manager

Agradecimentos

Quero formular o meu agradecimento a todos os que de algum modo contribuíram para a realização deste trabalho, nomeadamente:

- Ao meu orientador, Professor Doutor Paulo Sampaio, pela orientação, confiança transmitida, críticas e muitas sugestões práticas dadas ao longo de todo este período. Pela partilha deste interesse pela busca da “interacção natural” com essas máquinas a que chamamos computadores. Não me esquecerei, de modo algum, das vezes em que arrematou algumas afirmações com um concludente “Faz parte!”. É uma expressão que passou a “fazer parte” do meu léxico;
- À Universidade da Madeira e ao Departamento de Matemática e Engenharias, que tornaram possível, em boa hora, a realização do mestrado;
- À comissão responsável pela edição de 2004-2007 do Mestrado em Engenharia Informática;
- À minha mulher, Guida, pela paciência, apoio e confiança. «Sem ti teria desistido há muito tempo!»;
- Aos meus queridos filhos, Sofia, Margarida e Pedro, pelo carinho e verdadeira amizade, ou seja por todo o amor. «Vocês são os meus melhores amigos!»;
- Aos meus pais, Justina e Álvaro, pelos bons exemplos que sempre me deram e continuam a dar;
- Aos meus irmãos, Maria João, Zéca e Carlota, pela amizade e força anímica que sempre me deram;
- Aos colegas e professores de mestrado, por constituírem um excelente grupo;
- A todas as pessoas com quem entrei em contacto durante este período, em particular a Carl Burke (Phd), da MITRE Corporation, e a Staffan Larsson (Phd), da Universidade de Gotemburgo;
- Ao centro de Ciência e Tecnologia da Madeira (CITMA) pela bolsa de estudo concedida durante a fase lectiva do curso;
- A todos “aqueles” que partilhando a sua sabedoria possibilitam que cada um nós possa aprender sem ter de recomeçar sempre do “zero”.

Índice geral

Índice geral.....	xiv
Índice das tabelas.....	xviii
Índice das figuras	xix
Índice dos anexos.....	xxi
Capítulo 1 Introdução.....	2
1.1 Contexto	2
1.2 Motivação	4
1.3 Contribuições desta dissertação	5
1.4 Organização da dissertação	7
Capítulo 2 Enquadramento do tema e Estado da Arte	10
2.1 Introdução	10
2.1.1 A interação natural como objectivo	11
2.2 O diálogo e a sua caracterização.....	12
2.3 Sistemas de Diálogo Multimodais	15
2.3.1 Modelo conceptual de um SDMM	16
2.3.2 O factor humano	20
2.4 Áreas de aplicação dos Sistemas de Diálogo	20
2.5 Gestores de Diálogo	21
2.5.1 Responsabilidades e funções do GD	21
2.5.2 Modelo conceptual para o GD	22
2.6 Abordagens à gestão do diálogo	24
2.6.1 Abordagens baseadas em Máquinas de Estados Finitos (MEF)	24
2.6.2 Abordagens baseadas em Frames	25
2.6.3 Abordagens baseadas em Planos	26
2.6.4 Abordagens baseadas em Information State Update (ISU).....	26
2.6.5 Abordagens baseadas em Agentes Colaborativos	28
2.6.6 Análise das abordagens	29
2.7 Ferramentas para a modelação de diálogos	30
2.7.1 O CSLU Toolkit.....	31
2.7.2 O Loquendo Spoken Dialogue System	32
2.7.3 O Midiki	33
2.7.4 Análise das ferramentas	34
2.8 Um projecto de referência.....	36

2.9	Conclusões.....	37
Capítulo 3	O Gestor de Diálogo Midiki.....	40
3.1	Porque seleccionamos o Midiki	41
3.2	Estrutura do diálogo.....	42
3.2.1	Elementos que compõem o diálogo.....	42
3.2.2	O domínio.....	43
3.2.3	Planos	43
3.2.4	Dialogue moves.....	43
3.2.5	Estratégias	44
3.3	Arquitectura	44
3.3.1	Estrutura modular	46
3.4	O Information State (IS).....	47
3.5	Suporte a diferentes teorias de diálogo.....	47
3.5.1	A teoria de diálogo Questions Under Discussion (QUD)	47
3.6	Conclusão	49
Capítulo 4	Metodologia Proposta	50
4.1	Introdução	50
4.2	Componentes da modelação do diálogo.....	52
4.2.1	Representação de alto nível em XML	53
4.3	Apresentação e descrição dos passos da metodologia	54
4.3.1	Componentes declarativas (passos 1 a 9)	54
4.3.2	Especificação do Domínio	56
	• Passo 1 - Construção de planos.....	56
	• Passo 2 - Inicialização de atributos e de sinónimos.....	59
	• Passo 3 - Inicialização de questões.....	61
4.3.3	Especificação do Léxico	61
	• Passo 4 – Correspondência de output matches.....	62
	• Passo 5 - Correspondência de input matches.....	64
4.3.4	Especificação das Cells.....	66
	• Passo 6 –Declaração de <i>slots</i>	67
	• Passo 7 - Declaração de <i>queries</i>	68
	• Passo 8 - Declaração de <i>methods</i>	68
	• Passo 9 – Especificação de handlers	69
4.3.5	Componentes procedimentais (passos 10 a 13)	69

•	Passo 10 – Implementação de handlers	70
•	Passo 11 - Parametrização da classe Tasks	70
•	Passo 12 - Parametrização da classe DomainAgent	70
•	Passo 13 - Parametrização da classe “domain executive”	71
4.4	Conclusão	71
Capítulo 5 Implementação da Ferramenta de Autoria		74
5.1	Introdução	74
5.2	Descrição geral.....	75
5.3	Casos de uso.....	75
5.4	Arquitectura	80
5.4.1	Componentes da arquitectura.....	80
5.4.2	A base de dados que suporta a ferramenta	82
5.5	Interfaces de utilizador e ecrãs	84
5.6	Opções tecnológicas	103
5.7	Dialogue moves e estratégias suportados pela ferramenta.....	104
5.8	Ferramentas utilizadas no desenvolvimento da aplicação	105
5.9	Instalação do software.....	105
5.10	Conclusões.....	107
Capítulo 6 Caso de Estudo.....		108
6.1	Introdução	108
6.2	Requisitos e Casos de uso	109
6.3	Análise com vista à criação do diálogo/planos.....	111
6.4	Aplicação da metodologia a este caso.....	114
6.4.1	Especificação das componentes declarativas.....	115
6.4.2	Criação das componentes procedimentais	123
6.5	Geração de código e compilação.....	125
6.6	Execução do diálogo no Midiki	127
6.6.1	Correr o diálogo numa janela de texto	127
6.6.2	Um exemplo de interacção por fala com reconhecimento e síntese de fala em Português europeu.....	131
6.6.3	Debugging de um diálogo em runtime	132
6.7	Conclusão	132
Capítulo 7 Conclusões		134
7.1	Trabalhos futuros.....	135

<i>Publicações do autor.....</i>	<i>137</i>
<i>Anexos.....</i>	<i>149</i>

Índice das tabelas

Tabela 1: Comparação das diferentes abordagens à gestão do diálogo	29
Tabela 2: Comparação das três ferramentas (CSLU Toolkit, Loquendo SDS e Midiki) e respectivas abordagens para gestão do diálogo	35
Tabela 3: Os diferentes agentes que compõem o Midiki	45
Tabela 4: Lista de passos da metodologia e componentes declarativas em que se enquadram	55
Tabela 5: Tipos de <i>output matches</i> que podem ser definidos com a ferramenta.....	95
Tabela 6: Tipos de <i>input match</i> que podem ser definidos com a ferramenta.....	97
Tabela 7: Análise de requisitos genéricos e sua correspondência com os elementos da metodologia	111
Tabela 8: Correspondência dos requisitos genéricos com os elementos da metodologia para o caso de estudo	112
Tabela 9: <i>Input matches</i> definidos para o “serviço ATM”	120
Tabela 10: Descrição das estratégias suportadas e respectivos <i>dialogue moves</i> de saída.....	154
Tabela 11: Descrição dos <i>dialogue moves</i> de entrada (obtidos a partir dos <i>inputs</i> do utilizador)	157
Tabela 12: Tipos de construtores de entrada para os <i>output matches</i>	159
Tabela 13: Tipos de construtores de saída dos <i>output matches</i>	160
Tabela 14: Tipos de construtores de saída dos <i>input matches</i>	161
Tabela 15: Texto genérico a incluir nos <i>output matches</i>	162
Tabela 16: Texto genérico a incluir nos <i>input matches</i>	163
Tabela 17: Classes a invocar em <code>MidikiAuthoringGenerateJava.jar</code> para gerar as classes Java que especificam um diálogo no Midiki.....	165

Índice das figuras

Figura 1: Utilização de fala, gestos e expressões faciais no sistema SmartKom.....	4
Figura 2: Várias formas alternativas de <i>input</i> no sistema MATCH.....	11
Figura 3: Componentes de uma arquitectura genérica de um SDMM	17
Figura 4: Sistema de diálogo em arquitectura <i>pipeline</i>	19
Figura 5: Componentes genéricos de um GD	23
Figura 6: Ilustração da modelação de um subdiálogo por uma máquina de estados finitos	25
Figura 7: Ilustração de um <i>frame/form</i>	26
Figura 8: Estrutura de dados de um information state no Godis.....	28
Figura 9: Interfaces de trabalho do CSLU Toolkit.....	32
Figura 10: Componentes de um sistema desenvolvido com base no Loquendo SDS.....	33
Figura 11: Exemplo de classe Java do "domínio" de um diálogo e de um ecrã de edição na ferramenta web	34
Figura 12: Arquitectura (por agentes) do Midiki.....	45
Figura 13: Os diferentes elementos que compõem a parte declarativa da implementação de um diálogo	55
Figura 14: <i>Tags</i> XML para os elementos declarativos.....	55
Figura 15: Principais casos de uso para a ferramenta.	76
Figura 16: Decomposição/detalhe do caso de uso "criar projecto/diálogo"	77
Figura 17: Padrão MVC implementado	81
Figura 18: Componentes da ferramenta de engenharia/autoria de diálogos	81
Figura 19: Organização geral da interface da ferramenta	85
Figura 20: Lista de projectos de um autor	87
Figura 21: Criação/Edição de campos relativos a um projecto.....	88
Figura 22: Criação/Edição de elementos relativos a uma <i>cell</i>	88
Figura 23: Criação de novo plano - opção "NEW PLAN"	89
Figura 24: Edição de elementos que caracterizam um plano	89
Figura 25: Vista parcial de uma lista de estratégias de um plano, com destaque para a opção de inserção de novas estratégias.....	90
Figura 26: Criação/Edição de uma estratégia do tipo <i>Findout</i>	90
Figura 27: Criação/Edição uma estratégia do tipo <i>IfThen</i>	91
Figura 28: Ecrã de pesquisa/filtragem de estratégias.....	91
Figura 29: Criação/Edição de atributos	92
Figura 30: Inicialização de questões (criação de <i>question types</i>).....	93

Figura 31: Edição de <i>question types</i>	94
Figura 32: Inserção de <i>output matches</i>	94
Figura 33: Edição de <i>output matches</i>	96
Figura 34: Inserção de <i>input matches</i>	97
Figura 35: Edição de <i>input matches</i>	98
Figura 36: <i>Input match</i> para um <i>answerTask</i>	99
Figura 37: Menu de criação de <i>slots</i>	100
Figura 38: Menu de especificação de <i>cell queries</i>	101
Figura 39: Menu de especificação de <i>cell methods</i>	101
Figura 40: Menu de especificação de <i>handlers</i>	101
Figura 41: Ecrã de selecção de <i>cells e handlers</i> para geração	103
Figura 42: Casos de uso para a simulação a implementar	109
Figura 43: Campos que caracterizam um projecto	114
Figura 44: Campos que caracterizam uma <i>cell</i>	115
Figura 45: Estratégias do plano principal do diálogo (<i>ATMmainPlan</i>)	116
Figura 46: Estratégias do plano <i>consultarsaldoPlan</i>	116
Figura 47: Estratégias do plano <i>depositarPlan</i>	117
Figura 48: Definição do atributo “contas” e seus valores	118
Figura 49: Tipos de questões do diálogo	119
Figura 50: Definição do <i>output match</i> para <i>Findout(qualOperacaoPretendida)</i>	119
Figura 51: Definição do <i>output match</i> para <i>Inform(valSaldoConsulta)</i>	120
Figura 52: <i>Input match</i> que corresponde a um <i>answerTask</i>	121
Figura 53: <i>Slots</i> criados para o diálogo do caso de estudo	122
Figura 54: <i>Handlers</i> para as <i>queries</i> existentes	123
Figura 55: Estrutura de classes do Midiki integradas no <i>workspace</i> do Eclipse	126
Figura 56: Detalhe da estrutura de pastas do Midiki (pasta “java”)	126
Figura 57: Janela de interacção por texto em linguagem natural	128
Figura 58: Uma configuração para interacção por fala com os diálogos criados	131
Figura 59: Barra de opções do Philips FreeSpeech 2000	131
Figura 60: Estrutura de “classes” JET para o pacote <i>MidikiAuthoringGenerateJava</i> ..	166

Índice dos anexos

ANEXO I – XML Document Type Definitions (DTDs):

- i) DTD relativo à componente declarativa “domínio”;
- ii) DTD relativo à componente declarativa “léxico”;
- iii) DTD relativo à componente declarativa “cells”;

ANEXO II – Os *dialogue moves* e as estratégias no Midiki;

ANEXO III – Tipos de construtores relacionados com os *output matches* e os *input matches*;

ANEXO IV – A *Java Emitter Templates* (JET);

ANEXO V – Modelo relacional da BD da ferramenta.

Capítulo 1

Introdução

Neste capítulo apresentamos o contexto (secção 1.1), a motivação (secção 1.2), as contribuições (secção 1.3) e a organização desta dissertação (secção 1.4).

1.1 Contexto

Desde o início da era da informação que nos adaptamos aos condicionalismos e limitações impostos pelos diversos tipos de interfaces disponíveis, as quais têm sido essencialmente baseadas em linhas de comandos e interfaces gráficas do tipo WIMP (*Windows, Icons, Mouse, Pointer*). Continuamos a utilizar teclados *qwerty*, cujo *layout* foi concebido com o intuito de abrandar a velocidade de digitação das dactilógrafas, no tempo em que os mecanismos dos teclados não eram capazes de corresponder à habilidade das suas utilizadoras.

À medida que os equipamentos e os serviços evoluem, tendem a incorporar cada vez mais funcionalidades, apresentando-se cada mais vez mais completos e, por norma, mais complexos. A disponibilização de tais serviços ao utilizador (ex.: um utente ou cliente) implica muitas vezes que sejam apresentadas demasiadas opções de “menu”, com o utilizador a ter de filtrar as opções que pretende através de longas listas e hierarquias de sub-menus, muitas vezes apenas para aceder a duas ou três funcionalidades que usa e/ou necessita mais frequentemente. Não raramente, os menus e as interfaces de tais dispositivos e aplicações são concebidos sem ter em

devida conta o modelo mental do utilizador final, revelando desfasamentos entre o modelo mental de quem concebe as interfaces e o padrão de utilização de quem realmente as vai utilizar (ditando muitas vezes o sucesso ou o fracasso de muitos produtos no mercado). Se em vez disso bastar ao utilizar simplesmente identificar directamente, através da sua fala, uma tarefa ou serviço pretendidos, será certamente mais fácil [Branco *et al.*, 2005]; [Larsson, 2005]. E, caso o utilizador não seja capaz de indicar directamente o que pretende, poderá ser orientado pelo sistema através de um diálogo, permitindo-lhe continuar a alcançar o seu objectivo.

As pessoas comunicam entre si de muitas formas, constituindo a fala a principal forma (ou modo) utilizada. Se um sistema for capaz de dialogar com o utilizador, poderá compreender a sua “ordem”, poderá solicitar alguma clarificação ou poderá mesmo aceitar outras expressões e/ou palavras pelas quais o utilizador poderá activar ou executar determinada funcionalidade ou serviço. Um sistema capaz de dialogar com um humano, com o objectivo de desempenhar determinada função, chama-se Sistema de Diálogo (SD) [Allen *et al.*, 2001]; [Johnston *et al.*, 2001]; [McTear, 2002]; [Delgado & Araki, 2005]; [Cavedon, 2005]; [Danieli & Manca, 2005]; [Jurafsky & Martin, 2006]; [Trung, 2006]. Para determinadas situações, contextos ou tipos de pessoas, as soluções deste tipo constituem uma alternativa melhor (ou a única) às soluções já existentes (ex.: um condutor pode indicar que seja reproduzida uma determinada música sem que tenha de desviar a sua atenção da condução). Construir e disponibilizar sistemas que sejam capazes de dialogar de forma natural com o utilizador, em particular através da fala, ainda apresenta consideráveis dificuldades de ordem prática [McTear, 1998]; [Allen *et al.*, 2001], nomeadamente a dificuldade de reconhecer correctamente o que é dito pelo utilizador humano (ex.: muitas vezes as máquinas tem de ser treinadas para reconhecerem devidamente a voz de determinada pessoa ou então os equipamentos não dispõem das capacidades de processamento requeridas). Não obstante tais dificuldades, soluções para serviços com base em indicações do utilizador recorrendo a linguagem natural, nomeadamente na forma escrita (que apresenta menos dificuldades de implementação do que a interacção por fala), estão a tornar-se cada vez mais comuns, aliviando os utilizadores da necessidade de se adaptarem a interfaces específicas para cada tarefa [Vanderheiden *et al.*, 2005].

De uma forma mais abrangente, a interacção natural vai além da linguagem natural e da fala e implica a utilização de todas as formas de interacção utilizadas comumente entre pessoas (fala, gestos, posturas, olhar, tacto, emoções, contexto, ambiente físico, e outros). Sistemas que suportem vários destes modos de interacção dizem-se

Sistemas Multimodais [Delgado & Araki, 2005] e constituem uma área de intensa investigação, tanto a nível académico como a nível industrial.

1.2 Motivação

«From Ovid's statue of Pygmalion to Mary Shelley's Frankenstein, Cao Xue Qin's Divine Luminescent Stone-in-Waiting to Snow White's mirror, there is something deeply touching about creating something and then having a chat with it» [Jurafsky & Martin, 2006].

A motivação maior para este trabalho é poder caminhar um pouco na direcção de um estado da arte dos Sistemas de Diálogo (SD), no qual pessoas e máquinas poderão interagir da mesma forma que os humanos interagem entre si, nomeadamente: i) utilizando todos os sentidos; ii) tomando em consideração todo um contexto/histórico; iii) adaptando-se com relativa facilidade a novas situações e iv) aprendendo e assimilando continuamente novos conhecimentos. A interacção em linguagem natural, recorrendo a sistemas capazes de dialogar, pode tornar a sua utilização mais amigável, mais eficaz, mais eficiente e mais rápida. Impõe-se a criação de sistemas mais inteligentes (que os actuais), capazes de “entender” o utilizador, capazes de dialogar com o utilizador de forma natural e robusta¹. A Figura 1 ilustra o estado da arte actual nesta área.

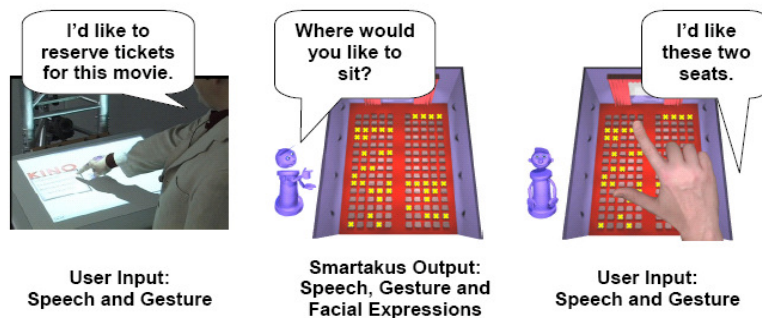


Figura 1: Utilização de fala, gestos e expressões faciais no sistema SmartKom

[Wahlster *et al.*, 2001b]

¹ Um SD robusto apresenta, entre outros, uma elevada taxa de reconhecimento de fala (idealmente de 100%); é capaz de interpretar correctamente cada intenção do interlocutor humano; é capaz de seguir uma estratégia de diálogo (ex. possuir capacidade de iniciativa e conseguir tratar adequadamente as interrupções); etc. Na Secção 2.9 apresentamos algumas das linhas de pesquisa neste sentido.

Os Sistemas de Diálogo incluem normalmente vários componentes, nomeadamente de entrada, de saída e de gestão do diálogo. Interessa-nos em particular esta última componente, o Gestor de Diálogo (GD), por ser a componente responsável por iniciar, executar e concluir um diálogo com o utilizador, por forma a que este obtenha rapidamente o que pretende, interagindo com o sistema do modo mais natural possível, recorrendo a uma “conversa” (um diálogo) com esse sistema, [Trung, 2006]; [Delgado & Araki, 2005]. Para realizar um diálogo, o GD precisa de incorporar as regras que o regem. O GD deve ainda incorporar um modelo da(s) tarefa(s) que poderá desempenhar, incluindo a especificação do conteúdo a apresentar ao utilizador (ex.: que texto vai apresentar ao utilizador para colocar-lhe questões ou apresentar-lhe resultados).

Relativamente aos gestores de diálogo disponíveis, pelo menos aqueles disponibilizados gratuitamente ou em regime de código aberto (alguns dos quais são apresentados na secção 2.7), verificamos que na maioria dos casos não apresentam metodologias claras para a modelação de diálogos e especificação de tarefas (ou não apresentam mesmo qualquer metodologia). O mesmo problema se apresenta relativamente às ferramentas e/ou funcionalidades que esses sistemas disponibilizam como suporte a essa modelação. As contribuições desta dissertação vão no sentido de colmatar tais lacunas, conforme descrevemos na secção 1.3.

1.3 Contribuições desta dissertação

Como referimos na secção 1.2, alguns dos gestores de diálogo são disponibilizados sem que sejam apresentadas metodologias e ferramentas claras e intuitivas para a modelação e autoria de diálogos. As nossas principais contribuições para esta área são a proposta e a implementação de: i) uma metodologia e ii) uma ferramenta para a modelação de diálogos [Quintal & Sampaio, 2007].

O Gestor de Diálogo (GD) que seleccionámos é um exemplo de um GD que, possuindo propriedades muito interessantes, é disponibilizado sem uma ferramenta de autoria e sem que seja apresentada uma metodologia clara e intuitiva. Na sua versão base, o MIDIKI [Burke *et al.*, 2003]; [Burke, 2005] obriga a que toda a modelação de um diálogo seja realizada através de classes Java [SUN, 1994]. Passemos então à apresentação das nossas contribuições:

i) A metodologia compõe-se de treze passos (descritos no capítulo 4). Os passos 1 a 9 correspondem à especificação das componentes declarativas do diálogo:

Os passos 1 a 3 correspondem à especificação do "Domínio":

- Passo 1 - Construção de planos;

- Passo 2 - Inicialização de atributos e declaração de sinónimos, e;
- Passo 3 - Inicialização de questões.

Os passos 4 e 5 correspondem especificação do "Léxico":

- Passo 4 – Correspondência de *output matches*, e;
- Passo 5 - Correspondência de *input matches*.

Os passos 6 a 9 correspondem especificação das "Cells":

- Passo 6 –Declaração de *slots*;
- Passo 7 - Declaração de *queries*, e;
- Passo 8 - Declaração de *methods*;
- Passo 9 – Especificação de *handlers*.

Os passos 10 a 13 correspondem à especificação das componentes procedimentais do diálogo:

- Passo 10 – Implementação de *handlers*;
- Passo 11 - Parametrização da classe *Tasks*;
- Passo 12 - Parametrização da classe *DomainAgent*, e;
- Passo 13 - Parametrização da classe "*domain executive*".

Como aspecto adicional à metodologia, é proposto um *Document Type Definition* (DTD) [W3Schools, 2007] para a representação em XML [Sperberg-McQueen & Thompson, 2000]; [Jung, 2000] dos componentes declarativos da modelação de um diálogo no Midiki. O respectivo DTD é apresentado no Anexo I. Esta DTD pode servir de referência em trabalhos futuros, com este ou outros GDs.

ii) A ferramenta de autoria implementada consiste numa aplicação *web* (descrita no capítulo 5), a qual permite a modelação/prototipagem de diálogos para o Midiki com base na metodologia proposta, abstraindo a modelação do diálogo da sua codificação final em Java e automatizando a geração de código (sempre que exequível). O código Java gerado é compilável com as restantes classes que compõem o sistema de gestão de diálogo no Midiki.

Com o intuito de testar a ferramenta (mesmo que informalmente), foi concebido e implementado um **caso de estudo**, criando um projecto totalmente de raiz. O caso de estudo (apresentado no Capítulo 6) consistiu numa simulação simplificada de uma interacção entre um cliente de um banco e uma caixa do tipo *Multibanco*. Interessava-nos sobretudo avaliar a facilidade acrescida na modelação de diálogos e testar o código Java gerado pela ferramenta.

Ainda como contributo desta dissertação, consideramos relevante a disponibilização de mais um documento desta natureza em Português, não só pelo seu valor intrínseco, mas também no contexto actual, em que se está a dar grande relevo, apoio e estímulo ao estudo dos sistemas de interacção em Português, em particular por fala [TecnoVoz, 2008]; [Microsoft, 2008].

1.4 Organização da dissertação

Este documento encontra-se organizado da seguinte forma:

- O segundo capítulo diz respeito ao “estado da arte”. São abordadas as seguintes temáticas: a interacção natural como objectivo; o diálogo e a sua caracterização; os sistemas de diálogo multimodais; áreas de aplicação dos sistemas de diálogo; os gestores de diálogo; as abordagens e ferramentas para a gestão do diálogo. O capítulo termina com a apresentação de um projecto de referência nesta área e alguns dos actuais desafios que se colocam neste domínio;
- No terceiro capítulo apresentamos o gestor de diálogo Midiki (nomeadamente porque seleccionamos o Midiki), a gestão do diálogo no Midiki e a sua arquitectura, incluindo uma descrição da estrutura do *information state*;
- No quarto capítulo descrevemos a metodologia proposta, designadamente os componentes da especificação do diálogo, a representação de alto nível em XML, a apresentação e descrição dos 13 passos da metodologia (componentes declarativas, nos passos 1 a 9, e componentes procedimentais, nos passos 10 a 13);
- No quinto capítulo descrevemos a ferramenta de autoria implementada, nomeadamente a sua descrição geral, os casos de uso, a arquitectura, a base de dados de suporte, as interfaces de utilizador e ecrãs, os elementos do diálogo suportados, as opções tecnológicas e a instalação do software;
- No capítulo seis apresentamos o nosso caso de estudo (“Acesso a um terminal multibanco”). São apresentados os requisitos, os casos de uso, a aplicação da metodologia ao caso de estudo (recorrendo à ferramenta de autoria), a especificação das componentes dos diálogo, a geração de código (na ferramenta) e a compilação e execução do diálogo no Midiki. Damos um exemplo de interacção por fala com o protótipo implementado recorrendo ao reconhecimento e síntese de fala em Português europeu;
- O sétimo, e último, capítulo diz respeito a considerações finais, nomeadamente sobre os resultados conseguidos e trabalhos futuros.

Esta dissertação termina com a lista de referências e inclusão dos anexos, que são os seguintes:

ANEXO I – XML Document Type Definitions (DTDs):

- i) DTD relativo à componente declarativa “domínio”;
- ii) DTD relativo à componente declarativa “léxico”;
- iii) DTD relativo à componente declarativa “cells”;

ANEXO II – Os *dialogue moves* e as estratégias no Midiki;

ANEXO III – Tipos de construtores relacionados com os *output matches* e os *input matches*;

ANEXO IV – A *Java Emitter Templates* (JET);

ANEXO V – Modelo Relacional da BD da ferramenta.

Capítulo 2

Enquadramento do tema e Estado da Arte

2.1 Introdução

Neste capítulo fazemos o levantamento do estado da arte relativamente aos Sistemas de Diálogo (SD) e em particular relativamente aos Gestores de Diálogo (GD) e às abordagens à gestão do diálogo.

O capítulo está estruturado da seguinte forma: após esta introdução é apresentado o conceito de diálogo e a sua caracterização, na secção 2.2; A secção 2.3 apresenta uma breve referência aos Sistemas de Diálogo Multimodais (SDMM); A secção 2.4 apresenta algumas áreas de aplicação dos Sistemas de Diálogo; A secção 2.5 introduz os Gestores de Diálogo; A secção 2.6 apresenta as principais abordagens à gestão do diálogo; Na secção 2.7 apresentamos algumas ferramentas de autoria; A secção 2.8 apresenta um projecto actual de referência. Por fim, a secção 2.9 apresenta algumas conclusões deste capítulo, nomeadamente alguns desafios actuais nesta área.

2.1.1 A interacção natural como objectivo

«In five years from now...I would like to see more people reaching the Web from devices big and small, fixed and mobile. I look forward to more voice technology--in hands-busy scenarios such as driving, and also to increase accessibility (e.g., for people with low vision).» [Berners-Lee, 2008].

Este desejo do director do *World Wide Web Consortium* (W3C), e inventor da Web, é demonstrativo da evolução actual na área da interacção Pessoa-Máquina que, cada vez menos obriga o utilizador a adaptar-se a interfaces específicas para cada tarefa. Soluções para serviços com base em indicações do utilizador recorrendo a linguagem natural estão a tornar-se cada vez mais comuns, assistindo-se a uma disponibilização crescente de interfaces suportando o toque, os gestos e a fala, definindo novos paradigmas para as interfaces Pessoa-Máquina. A Figura 2 ilustra como um mesmo pedido (saber quais os restaurantes italianos baratos em determinada zona) pode ser expresso de diferentes formas, através de: i) fala, ii) fala + indicação gráfica ou iii) apenas por indicação gráfica.

Fala: “show inexpensive italian restaurants in chelsea”

Multimodal: “cheap italian restaurants in this area”

Pen:

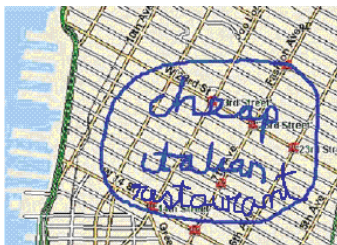


Figura 2: Várias formas alternativas de *input*² no sistema MATCH

(adaptado de [Johnston *et al.*, 2001])

O recente sucesso comercial de produtos como o *Apple iPhone*TM ou a consola de jogos *Nintendo Wii*TM é bem demonstrativo das vantagens e potencialidades dos sistemas que suportam novas formas de interacção, cada vez mais próximas da interacção natural [Vanderheiden *et al.*, 2005].

² Utilizaremos o termo *input* ou *entrada* para nos referirmos ao fornecimento de informação de entrada, de um utilizador humano para um sistema (de diálogo). Da mesma forma utilizaremos o termo *output* ou *saída* para nos referirmos à apresentação de informação de saída do sistema para o utilizador humano.

As pessoas no seu ambiente diário utilizam vários canais de comunicação (sentidos) para comunicarem entre si (ex: voz e fala, reconhecimento de gestos/movimentos corporais, expressões faciais ou movimento dos olhos, tacto, etc). A interação entre pessoas e máquinas, de uma forma natural, utilizando os mesmos modos (ou modalidades) e sentidos que usamos na comunicação usual de pessoa para pessoa (e que inclui a fala, gestos, postura, etc) diz-se interação multimodal [Delgado & Araki, 2005]. Existe uma analogia entre os diferentes sentidos dos humanos (e de forma mais genérica, com as diferentes formas que os humanos utilizam para comunicar) e as diferentes modalidades num sistema de diálogo. A interacção Pessoa-Pessoa é multimodal por natureza. Os sistemas computacionais que implementam esse tipo de interação dizem-se Sistemas de Diálogo Multimodais (SDMM).

De entre as diferentes modalidades, é consensual que a fala (e a utilização da voz em geral) constitui a forma principal pela qual os humanos se expressam, da mesma forma que a visão e a audição constituem os principais modos de recepção de informação dos humanos. De uma forma geral, a indicação de instruções ou pedidos através da fala são mais naturais para o utilizador, enquanto a recepção, numa mistura de fala e imagem é em geral mais eficiente.

«A linguagem em si tem sido desde sempre a marca de humanidade e sensibilidade, e a conversação ou diálogo de conversação é a arena mais fundamental e especialmente privilegiada da linguagem. O diálogo é certamente o primeiro tipo de linguagem que aprendemos quando crianças e, para a maioria de nós, é o tipo de linguagem em que mais comumente nos envolvemos, quer estejamos a preparar o almoço ou a comprar selos postais, participando em reuniões empresariais ou a falar com as nossas famílias, reservando viagens ou reclamando sobre as condições meteorológicas.» [Jurafsky & Martin, 2006].

Na secção seguinte fazemos uma caracterização do diálogo no sentido de identificarmos os requisitos a que um SD deve obedecer.

2.2 O diálogo e a sua caracterização

Um sentido comum do termo “dialogar” tem a ver com a resolução de problemas. Praticamente todas as actividades humanas envolvem dialogar com outras pessoas como forma de levar a cabo as mais variadas acções.

Uma definição de **diálogo** num dicionário de referência da língua portuguesa [Infopedia, 2007] indica que se trata de um substantivo masculino, com o seguinte significado:

1. Conversa entre duas ou mais pessoas;

2. Troca de ideias para se chegar a um entendimento;
3. Obra literária ou científica em forma de conversação;
4. Alternância de dois factores complementares um do outro.

Um diálogo implica uma sequência de troca de informação que se mantém coerente ao longo do tempo e por definição implica uma “conversa”, entre dois ou mais interlocutores, uma troca de “ideias” com vista a um entendimento e uma alternância entre emissor e receptor [Oviatt, 2002]; [Trung, 2006]. O diálogo é um fenómeno social (possui regras) e um diálogo eficaz deve ser colaborativo. Se quisermos criar máquinas capazes de dialogar com pessoas, teremos de implementar sistemas capazes de “conversar”, ie, de trocar ideias, em alternância, e que progridam com vista a um entendimento e um resultado final. A criação de sistemas com essas capacidades apresenta desafios consideráveis. De acordo com James Allen, em [Allen *et al.*, 2001], os quatro principais desafios que se colocam são: i) gerir a complexidade da linguagem (natural) a utilizar, associada à tarefa; ii) integrar o SD com o sistema de *back-end* que disponibiliza o serviço pretendido; iii) a necessidade de reconhecimento de uma intenção (por parte do utilizador humano) como aspecto chave do processo de entendimento, e; iv) suportar e possibilitar a iniciativa mista, onde tanto o sistema como o utilizador podem liderar a iniciativa do diálogo em diferentes alturas, tornando a interacção mais natural, mais eficiente e mais eficaz.

Embora para os humanos o diálogo (falado) seja a forma mais natural de comunicação, este inclui características que o tornam bastante complexo para sistemas computacionais, pelo que ainda não existem sistemas de diálogo robustos mesmo quando se considera apenas a fala isoladamente [Pallotta & Ballim, 2001]; [Oviatt, 2002]. Um diálogo efectivo envolve não apenas a fala (o que em si já introduz vários desafios, nomeadamente quando consideramos a “linguagem natural”), mas também outras modalidades:

- **Gestos:** possuem um papel crucial na comunicação (ex.: apontar, acenar, desenhar, escrever, clicar, etc);
- **O olhar e os sons vocais em geral,** que também desempenham um papel importante numa comunicação efectiva.

Para a modelação de um diálogo, é necessário caracterizar as propriedades e as regras que definem o diálogo. Para isso, coloca-se a questão de saber que recursos e conhecimentos serão necessários para essa modelação, ou seja, que aspectos são relevantes para uma/a gestão do diálogo? Seguidamente apresentamos alguns dos aspectos que deverão ser de algum modo considerados na criação de um sistema de

diálogo [Allen *et al.*, 2001]; [McTear, 2002]; [Armstrong *et al.*, 2003]; [Melichar, 2005]; [Jurafsky & Martin, 2006]:

- **Reconhecimento de fala:** Os erros de reconhecimento são bastante comuns (ex. devido a interferência provocada pelo ruído);
- **Interpretação** (depende do contexto): Como reconhecer a intenção de cada contribuição para o diálogo? Implica a necessidade de revisões e correcções:
 - Elipses e fragmentos: as pessoas utilizam frequentemente frases parciais por forma a evitarem a repetição. A informação em falta tem de ser reconstruída a partir do contexto do diálogo;
 - Referências: palavras como “este / aquele / eu / ele / aquilo” apenas podem possuir significado se interpretadas em determinado contexto, e;
 - Significado indirecto: muitas expressões não possuem o seu significado literal.
- **Estratégias de diálogo:** Como determinar a “sua vez de falar”? Como garantir que há um entendimento comum / partilhado entre as partes?:
 - Vez de falar: no diálogo entre pessoas, a sobreposição é normalmente imperceptível e os compassos de espera entre cada vocalização³ são muito reduzido (menos de 1/10 de segundo);
 - Desambiguação de *inputs* do utilizador: as expressões naturais são muitas vezes ambíguas, pelo que o sistema deve implementar mecanismos para desambiguação de *inputs* do utilizador. A desambiguação pode ser implícita (deduzida do contexto) ou explícita (perguntando ao utilizador);
 - Confirmação: o sistema deve requerer ao utilizador a confirmação de informação que tenha sido obtida com um grau de confiança demasiado baixo ou que não se enquadre no contexto do diálogo;
 - Obtenção da informação em falta: o sistema deve tomar a iniciativa de obter a informação em falta de forma a completar a tarefa com sucesso;

³ Tradução de “*utterance*”: Significa enunciado, elocução, fala ou expressão oral. Implica a utilização de sons em geral para comunicação, normalmente palavras mas também outros sons (consulta no Wordnet [Miller, 2006]). Em vez do termo vocalização também poderíamos dizer expressão oral (ou simplesmente expressão). Ao longo deste documento utilizaremos o termo “expressão” para nos referirmos a um *input* do utilizador, mesmo quando não corresponder apenas a uma vocalização e/ou ao seu texto, ie, mesmo que esse *input* incorpore outras contribuições, para além da fala (ex. fala + gestos).

- Detecção de confusão ou erro por parte do utilizador: uma vez que os sistemas de diálogo são imperfeitos, é importante implementar diferentes estratégias que permitam resolver os problemas que surgem na comunicação. Por ex.: se o utilizador não sabe como responder a uma questão, o sistema deve ajudá-lo. A indicação de confusão do utilizador pode ser detectada por ex.: se afirmar “Não sei” ou através de pistas audiovisuais. Nestas situações assiste-se a um não progresso do diálogo (e isso deve ser evitado), e;
- Descrição de acções semânticas que não seriam visíveis de outra forma: o sistema deve notificar sempre o utilizador nos casos em que assuma alguma coisa de forma implícita para a qual não tenha sido clara a concordância do utilizador.
- **Iniciativa:** Quem lidera a iniciativa? (o utilizador, o sistema, iniciativa mista);
- **Interrupções:** Se o diálogo for interrompido, será que se manteve o assunto/tópico? Há necessidade de alterar o tópico ou concluir o diálogo?
 - Alteração de tópico: o sistema deve ser capaz de reconhecer quando o utilizador sai do contexto. Se possível, o sistema deve ser capaz de mudar para o novo contexto, ou então trazer o utilizador de novo para o contexto relevante/anterior;
- **Personalização da interacção:** Com base no histórico do utilizador? Com base no histórico do diálogo? Com base no perfil do utilizador?

Um sistema interactivo que suporte de forma robusta e completa todos os requisitos anteriores aproximar-se-á bastante daquilo que consideramos interacção natural, que corresponde à interacção usual Pessoa-Pessoa.

2.3 Sistemas de Diálogo Multimodais

A evolução das interfaces passa pela integração, nas técnicas de interacção, das capacidades pré-adquiridas e usadas diariamente por qualquer pessoa, como por exemplo: a fala, a escrita e os gestos. É neste sentido que surge o conceito de interface multimodal, como uma forma de capacitar uma máquina para a interacção natural com uma pessoa. Ao contrário das habituais interfaces WIMP, as interfaces multimodais incluem normalmente um conjunto de modalidades tais como voz e gestos, para a entrada de dados, e informação multimédia sincronizada para a saída [Beckham *et al.*, 2001]. Essas tecnologias apresentam uma visão de uma geração de ferramentas e equipamentos que permitirão às pessoas dispor do tipo de interacção/interface mais conveniente em cada contexto, de forma robusta.

Cada modalidade de interacção utilizada pode ser vista como um canal de comunicação entre um utilizador e a máquina. Estes modos ou modalidades podem ser combinados numa interface de um sistema de diálogo que seja capaz de gerir/manipular várias modalidades. Um sistema de diálogo assim criado é um Sistema de Diálogo Multimodal [Larson & Candell, 2003]; [Delgado & Araki, 2005]; [Rudnicky, 2005]; [Kellner, 2005]; [Trung, 2006]. A integração de múltiplas modalidades de entrada permite alcançar uma maior expressividade, pelo recurso a fontes de informação complementares; Permite também maior fiabilidade ao conteúdo devido à utilização de modalidades redundantes [Melichar, 2005].

Ao nível da entrada, algumas modalidades são mais eficientes que outras para certos tipos de conteúdos, tarefas, utilizadores ou contextos (ex.: cliques de rato ou toques podem compensar limitações na tecnologia de reconhecimento de voz). Através de uma distribuição optimizada pelas diferentes modalidades numa única entrada é possível uma codificação mais eficiente de informação de um *input* (do utilizador). Ao nível da saída, podem ser adoptadas estratégias de diálogo que visem a utilização da forma mais conveniente de apresentação de *output* (ex.: confirmações vocais longas podem ser substituídas de forma favorável por curtos indicadores visuais). Tanto ao nível da entrada como da saída, a multimodalidade permite uma melhor adaptação às necessidades da situação, tarefa, e/ou preferências do utilizador, permitindo uma interacção conveniente, eficiente e segura. A redundância permite um reconhecimento e entendimento mais fáceis, mais robustos e fiáveis, tanto da parte do sistema como da parte do utilizador (através de modalidades concorrentes).

No seu conjunto, a multimodalidade permite e pretende aumentar a velocidade, a precisão e a naturalidade da interacção Pessoa-Máquina [Melichar, 2005]. Uma eficiência acrescida constitui a principal vantagem dos sistemas multimodais.

Um modelo de um SD inclui como componente central um gestor de diálogo, com o qual interagem os restantes componentes. Na secção seguinte apresentamos um modelo conceptual para um SD multimodal (SDMM) e descrevemos os seus componentes.

2.3.1 Modelo conceptual de um SDMM

A Figura 3 apresentada um modelo de componentes genéricos principais de uma arquitectura de um sistema de diálogo, com o gestor de diálogo como componente central.

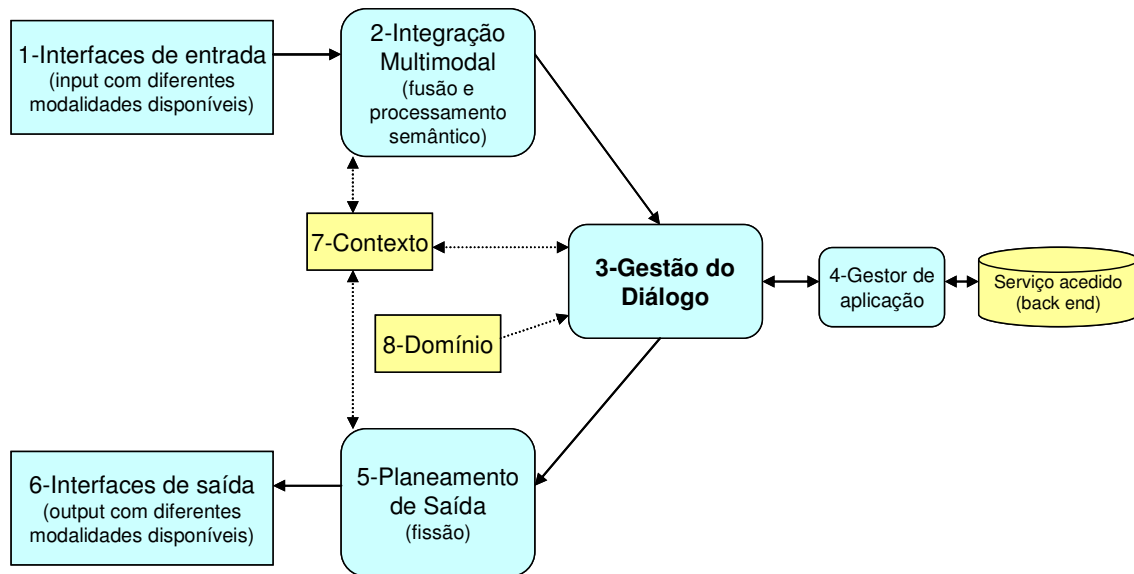


Figura 3: Componentes de uma arquitectura genérica de um SDMM

Os componentes do modelo conceptual apresentado na Figura 3, são apresentados e discutidos a seguir [McTear, 2002]; [Larson & Candell, 2003]; [Carbonell, 2005]; [Delgado & Araki, 2005]; [Kellner, 2005]; [Rudnický, 2005]; [Trung, 2006]; [Melichar, 2008]:

1) **Interface(s) de entrada:** são responsáveis por captar os *inputs* do utilizador, nas modalidades disponíveis (por ex. o reconhecimento de fala transforma o input de voz humana (fala) numa lista de transcrições textuais);

2) **Integração ou fusão multimodal:** componente que funde os diversos *inputs* das diferentes modalidades de entrada activas. Pode incluir processamento semântico desse *input*. Ao mapeamento de diferentes canais de entrada para um único fluxo semântico chamamos fusão ou integração multimodal. O processamento semântico extrai o conhecimento das vocalizações feitas (ex. texto) e pode ser realizado antes ou depois da fusão. O conhecimento (a semântica associada ao texto produzido) é expresso numa representação formal que seja entendida por um gestor de diálogo. Dependendo das técnicas utilizadas, este processo pode envolver vários subprocessos, ex.: análise sintáctica, análise semântica, etc;

3) **Gestão do diálogo:** componente central do sistema, responsável por decidir qual a saída a gerar, tendo em conta o estado do sistema e o *input* actual, resultando num novo estado do sistema. O Gestor de diálogo (ou gestor de interacção) é a componente central que controla o fluxo da interacção com o utilizador, baseado num determinado modelo de diálogo. O *input* para este processo é uma representação

formal da vocalização do utilizador e o output é uma mensagem formal a ser gerada como resposta. O gestor de diálogo também pode recorrer ou integrar informação de suporte (ex. informação de contexto) e aceder a serviços externos através de um gestor de aplicação. O gestor de diálogo tem como função/papel direccionar a conversa através de um caminho produtivo (no sentido do objectivo identificado), tornando a interacção com o utilizador tão natural e eficiente quanto possível e, ao mesmo tempo, mantendo o utilizador nas fronteiras do domínio [Goddeau *et al.*, 1996];

4) **Gestor de aplicação:** encapsula e trata das operações específicas da aplicações externas (ex.: comunicação com um SGBD). O gestor de diálogo deve operar a um nível genérico, independente do domínio, enquanto o gestor de aplicação trata dos aspectos relacionados com o serviço acedido;

5) **Planeamento de saída:** a partir da resposta gerada pelo GD, este componente é responsável por construir a saída do sistema, seleccionando o(s) modo(s) de saída apropriados. Constrói a forma da mensagem a ser comunicada ao utilizador com base na mensagem formal gerada pelo gestor de diálogo. Define qual a utilização das modalidades de saída disponíveis e qual a proporção adequada do *output* a colocar em cada modalidade. Ao mapeamento do fluxo semântico de saída para vários canais de saída chamamos *fissão*;

6) **Interface de saída:** constituem as interface(s) física(s) que transmitem o *output* ao utilizador, e;

7) e 8) **O Contexto e o Domínio** constituem elementos de suporte (ou fontes de conhecimento) para o GD. Além de informação específica sobre o tópico ou tópicos suportados, o *Domínio* contém normalmente, informação sobre como o sistema deve proceder (ex.: um plano). O *Contexto* contém informação dinâmica sobre o ambiente e os intervenientes (regista a estrutura do diálogo e permite referências contextuais). Estes dois elementos podem estar ou não integrados na componente GD.

A forma mais simples de organização e implementação dos componentes de um sistema de diálogo é a arquitectura em *pipeline* indicada na Figura 4 (na prática estes componentes funcionam normalmente em arquitectura distribuída).

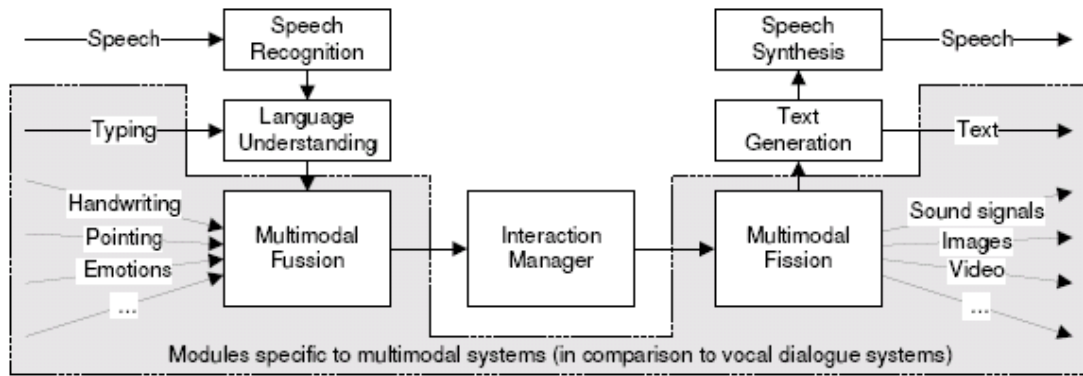


Figura 4: Sistema de diálogo em arquitectura *pipeline*

[Delgado & Araki, 2005]

A construção e integração do sistema de diálogo serão facilitadas se os diferentes módulos forem compatíveis com determinada plataforma de software (*framework*), a qual deve apresentar as regras de desenho dos módulos e oferecer a infraestrutura de comunicação necessária entre módulos. As *frameworks* existentes que respondem de forma satisfatória a estes requisitos são a *Galaxy Communicator II* [Seneff *et al.*, 1998]; [MITRE, 2003] e a *Open Agent Architecture* (OAA) [Martin *et al.*, 1999].

Em particular as tecnologias de reconhecimento de voz e síntese de fala, já se encontram razoavelmente maduras e podem, em geral, ser tratadas através do paradigma da “caixas preta” na concepção de sistemas multimodais. Estes componentes podem ser adquiridos a terceiros e integrados no sistema.

Apesar de não existirem ainda standards universais para a representação de informação multimodal e para a modelação/especificação de diálogos, existem vários standards que são utilizados e suportados de forma generalizada em determinadas áreas, facilitando a integração de componentes num SD. Está a ser feito desde há algum tempo um esforço em termos de standardização destas tecnologias, com particular relevo para o VoiceXML [VoiceXML, 2004]. O VoiceXML é um standard W3C para aplicações de voz. Constitui o standard de base da maioria dos sistemas comerciais actuais, nomeadamente em serviços acedidos por telefone.

Outras tecnologias relevantes mas menos universais incluem a SALT [Wang, 2002], a XHTML+Voice [IBM, 2004], a EMMA [W3C, 2003] e a DAMSL [Core, 1997]. A este propósito consultar também [Larson, 2005] ou [Delgado & Araki, 2005], onde são abordadas algumas linguagens e tecnologias para o desenvolvimento de aplicações multimodais.

2.3.2 O factor humano

A multimodalidade por si só não garante um aumento da qualidade de uma interface [Kvale *et al.*, 2003]. É necessário que haja uma adequada integração entre diferentes modalidades. Uma integração correcta das várias modalidades passará por dar ao sistema e/ou ao utilizador a capacidade de escolher as modalidades de entrada e/ou saída mais apropriadas para cada situação e a capacidade de reconhecer expressões de entrada construídas com informação proveniente de diferentes modalidades (possibilidade de passarmos mais e melhor informação para o sistema, com menor esforço).

O sistema pode e deve, sempre que possível, incluir um modelo do utilizador, o qual inclui assumpções explícitas sobre todos os aspectos relacionados com o utilizador que possam ser relevantes para o comportamento do sistema no diálogo. Isso permite acelerar o fluxo do diálogo e aumentar a satisfação do utilizador ao interagir com o sistema.

2.4 Áreas de aplicação dos Sistemas de Diálogo

Sempre que o utilizador está a utilizar um SD, no sentido de encontrar uma solução/resposta a um problema ou obter uma informação que pretende, o sistema deverá direccioná-lo para o recurso adequado à resolução do seu problema, da forma o mais rápida e simples possível. A utilização de um sistema de diálogo que seja capaz de atender de forma eficiente o utilizador apresenta vantagens, tanto para o prestador do serviço como para o utente, tais como disponibilidade aumentada (ex.: 24 horas por dia) e custos reduzidos (porque será um sistema informático e não um recurso humano que realizará o trabalho).

A utilização de sistemas multimodais tem interesse para o utilizador em geral, mas reveste-se de particular interesse em alguns sectores, contextos e para alguns tipos de utilizadores, designadamente:

- Sistemas conversacionais para acesso a sistemas de informação, nomeadamente na área dos transportes (ex. informação de horários de viagens), da banca, da meteorologia, etc. Inclui vários tipos de serviços, em particular os serviços prestados por telefone, com atendimento automático;
- Colaboração/interacção com robots (em ambiente dinâmicos e imprevisíveis, funcionando em tempo real);
- Ambientes inteligentes (ex. em casa ou no carro, com personalização e altamente contextualizado), nomeadamente no apoio a pessoas com

deficiência ou com capacidade reduzida. Outra aplicação com elevado potencial é a da gestão de reuniões físicas (diálogo múltiplo, reconhecendo gestos, movimentos, escrita, desenho e fala);

- Sistemas para transcrição, interpretação, tradução e/ou resumo de documentos (ex. emails, relatórios, etc) e de discurso falado (ex. em reuniões, aulas, rádio e televisão, etc);
- Em toda a industria dos jogos interactivos;
- Tarefas complexas em ambientes dinâmicos (ex. manutenção aeroespacial), e;
- Ambientes de tutoria (sistemas de aprendizagem inteligentes, que envolvam e encaminhem o utilizador).

Concluimos aqui esta abordagem geral e introdutória aos sistemas de diálogo. Na secção 2.5 vamos concentrar a nossa atenção no componente central de um SD: o Gestor de Diálogo (GD).

2.5 Gestores de Diálogo

No nosso contexto, um gestor de diálogo é um sistema de software que permite gerir uma conversação entre um utilizador e uma máquina, normalmente em linguagem natural, numa ou mais modalidades [Traum & Larsson, 2000]; [Burke *et al.*, 2003]; [Buckley & Benzmüller, 2005]; [O'Neill *et al.*, 2005]; [Jurafsky & Martin, 2006]; [Wilks *et al.*, 2006]; [Quintal & Sampaio, 2007]. Um diálogo é normalmente modelado para determinado domínio de conhecimento e é associado ao tipo de tarefa (ou tarefas) a suportar.

Um GD é o componente central num SD pois controla o fluxo e a progressão da interação com o utilizador, com base num determinado modelo de diálogo. A entrada para o GD é uma representação formal das expressões do utilizador (expressas numa ou mais modalidades). A saída do GD será também uma representação formal do mesmo tipo, que é então convertida numa forma legível para o utilizador humano (em texto, fala, imagens ou outras modalidades). O GD mantém uma representação do estado do diálogo, gerindo o seu progresso e actualizando esse mesmo estado. A representação do estado do diálogo, e dos mecanismos que determinam o seu progresso, pode ser modelada e concretizada de diversas formas, como veremos na secção 2.6 (Abordagens à gestão do diálogo).

2.5.1 Responsabilidades e funções do GD

Integrado num SD, um gestor de diálogo desempenha normalmente um papel de mediador conversacional entre o utilizador e uma aplicação final, à qual o utilizador

accede dialogando com o GD. Uma interface conversacional dá ao utilizador a possibilidade de expressar o que pretende nos seus próprios termos, tal como o faria com outra pessoa, cabendo ao sistema lidar com a complexidade de dialogar com o utilizador [Seneff *et al.*, 1998]; [Quesada *et al.*, 2000]; [Allen *et al.*, 2001]; [Allen *et al.*, 2001b]. O gestor de diálogo tem de: i) determinar se, e quando, foi obtida informação suficiente do utilizador que permita a comunicação com a aplicação final (ou *back end*), e; ii) gerir a comunicação com essa aplicação externa e devolver os resultados ao utilizador. Em resumo, o gestor de diálogo recebe um *input*, processa-o (num determinado contexto/estado), determina uma resposta e passa o seu *output* ao “planeamento de saída” [Robinson *et al.*, 2004]. O gestor de diálogo é claramente um componente crítico de um sistema de diálogo.

Segundo Traum [Traum *et al.*, 1999], e Larsson [Traum & Larsson, 2003], as principais tarefas, ou responsabilidades, do GD são:

- Actualização do contexto do diálogo com base na interpretação da comunicação efectuada (e em curso);
- Realização da interface com o serviço acedido (ex.: base de dados), com o fim de coordenar o comportamento de diálogo e o raciocínio do sistema, e;
- Indicação de qual passo seguinte do sistema, de acordo com determinada estratégia de progresso implementada.

Na secção 2.5.2 fazemos uma análise do GD como um sistema, o qual integra um conjunto de modelos (ou fontes de conhecimento) que lhe permitem exhibir um leque de competências aceitáveis e esperadas num diálogo em determinado domínio.

2.5.2 Modelo conceptual para o GD

Um gestor de diálogo é um componente de um sistema de diálogo, mas é em si um sistema, divisível em vários componentes. Nesses componentes podemos identificar um núcleo do GD no qual se encontram as regras e a lógica que implementam determinado modelo de diálogo.

O núcleo do gestor de diálogo pode recorrer a um conjunto de fontes, as quais podem constituir componentes do GD, integrando-o como um todo [McTear, 2002]; [Melichar, 2008]; [Trung, 2006]. Colectivamente, permitem ao núcleo do GD representar e gerir o progresso do diálogo de forma coerente. Essas fontes podem conter regras de conversação; podem conter informações sobre o mundo ou sobre um determinado domínio de conhecimento; podem conter informações sobre um utilizador; podem conter um modelo da tarefa a executar, e; podem guardar informação sobre o histórico do diálogo; etc. Podemos englobar todas essas fontes no todo do GD, acrescentando-lhe mais ou menos competências, mais ou menos robustez e mais ou menos

complexidade. Este modelo conceptual é ilustrado pela Figura 5, baseado em [Wilson, 1991], na qual são indicadas as fontes a que o GD recorre.

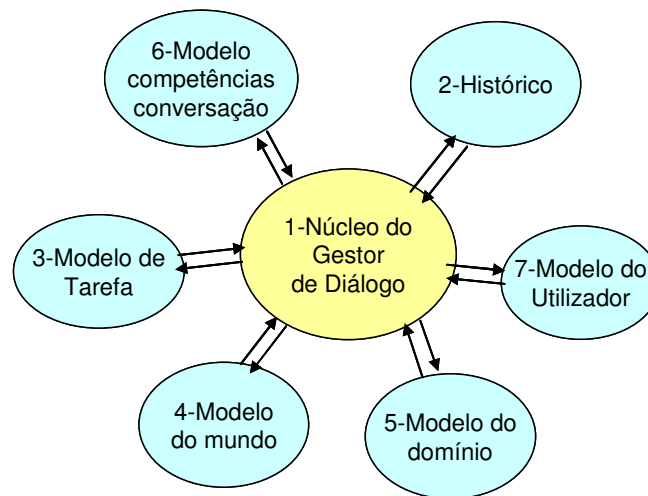


Figura 5: Componentes genéricos de um GD

Um modelo do gestor de diálogo pode incluir estas fontes de conhecimento [McTear, 2002]; [Trung, 2006], que contribuem para a gestão do diálogo e incluem:

- 1 **Núcleo do GD:** contém os algoritmos de controlo e as regras que determinam o comportamento do GD, de acordo com o modelo/teoria de diálogo implementada;
- 2 **Um histórico do diálogo:** regista o traço do diálogo, em termos das proposições já discutidas e das entidades já mencionadas. Esta representação fornece a base para uma coerência conceptual ao longo da conversação e para a resolução de anáforas e elipses;
- 3 **Modelo de tarefa:** uma representação da informação a ser obtida durante o diálogo e da forma como essa informação deve ser obtida (ex.: o plano do diálogo). Este registo é usado para determinar que informação ainda não foi obtida;
- 4 **Um modelo do mundo:** Este modelo contém informação geral de suporte a raciocínio de senso comum, se requerido pelo sistema;
- 5 **Um modelo do domínio:** um modelo com informação específica sobre o domínio de aplicação em consideração;
- 6 **Um modelo genérico de competências de conversação:** inclui princípios sobre algumas regras de participação num diálogo cooperativo (por ex.: no sentido do “princípio de cooperação”, de H. P. Grice [Infopedia, 2008]), e;
- 7 **Um modelo de utilizador:** este modelo pode conter informação relativamente estável sobre o utilizador, a qual pode ser relevante para o diálogo, bem como informação que mude no decurso do diálogo.

Cada implementação concreta de um GD poderá integrar, parcial ou totalmente, estes componentes de diferentes formas, mais ou menos explícitas. Na próxima secção caracterizamos as principais abordagens à gestão do diálogo.

2.6 Abordagens à gestão do diálogo

A abordagem à gestão do diálogo tem a ver com a forma como o estado do diálogo será representado, como as partes do discurso do interlocutor humano serão interpretadas e como o GD irá interagir/reagir, fazendo progredir o diálogo e actualizando o seu estado em cada passo.

Existe a necessidade de considerar e modelar vários elementos que irão integrar o gestor de diálogo:

- Uma estrutura de dados que permita representar o estado do diálogo;
- Uma forma de modelar a interacção esperada entre o interlocutor humano e o sistema;
- Uma forma de identificar e interpretar as sucessivas contribuições do utilizador, e;
- Uma dinâmica da gestão do diálogo, que defina como se passa de um estado para outro e permita especificar como responder ao utilizador.

A implementação destes elementos em sistemas informáticos implica que se recorra a linguagens de programação, estruturas de dados e *hardware* disponíveis. Passar dos modelos linguísticos para os modelos informáticos que os concretizam em sistemas práticos concretos viáveis leva a que se tenha muitas vezes de optar por soluções tecnológicas limitadas na prática, mas de complexidade controlável. Uma das primeiras abordagens aplicadas à gestão de diálogos consistiu na utilização de Máquinas de Estados Finitos (MEF) [Wikipedia, 2006]. Sistemas mais complexos têm sido sucessivamente criados, cada qual com as suas vantagens e desvantagens (como regra geral, um sistema mais flexível tende a ser menos robusto, e vice-versa). Na secção 2.5.3 apresentamos as abordagens mais significativas/tradicionais aplicadas à gestão do diálogo.

2.6.1 Abordagens baseadas em Máquinas de Estados Finitos (MEF)

Uma abordagem tradicional à gestão do diálogo baseia-se no modelo de estados, na qual o fluxo do diálogo é representado por um caminho através de uma MEF [McTear, 1998]. A estrutura do diálogo é representada por uma máquina de estados, na qual os

nós representam as questões/expressões (*prompts*) do sistema e cada transição entre nós corresponde a uma possível resposta/contribuição do utilizador a uma questão do sistema [Delgado & Araki, 2005]. Todas as variantes do diálogo são codificadas de forma rígida. Todos os caminhos possíveis são pré-determinados, ie, o fluxo do diálogo é rígido, e a iniciativa do diálogo é liderada pelo sistema. Estes modelos são adequados a situações em que a quantidade de possíveis estados é relativamente pequena, em que a iniciativa parte sempre do sistema e as respostas esperadas do utilizador são simples e curtas. Não são, assim, apropriados para sistemas que devam suportar linguagem natural (a comunicação humana expressa por fala ou texto usuais, sem restrições de forma ou de conteúdo). A Figura 6 ilustra uma representação de uma MEF para um (sub)diálogo codificado através de estados e transições.

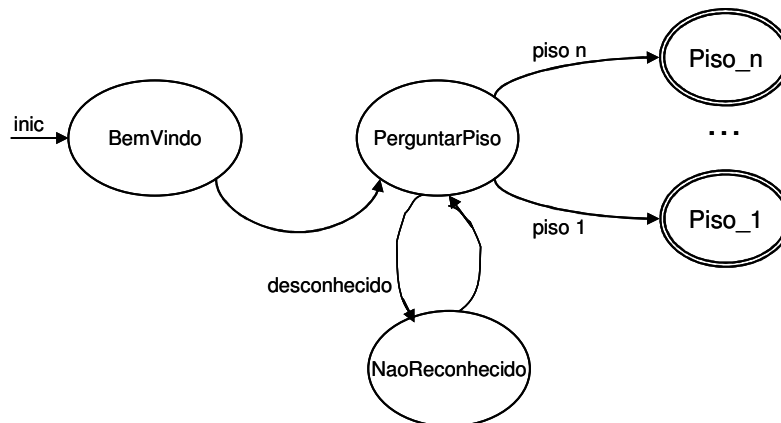


Figura 6: Ilustração da modelação de um subdiálogo por uma máquina de estados finitos
(adaptado de [Kordoni *et al.*, 2007])

2.6.2 Abordagens baseadas em *Frames*

Os algoritmos de planeamento baseados em *frames* (ou *slots*) funcionam de forma análoga ao preenchimento de um formulário electrónico (ou *form*) [Goddeau *et al.*, 1996]. Uma *frame* é composta por vários *slots*, nos quais a informação será representada (uma parte em cada *slot*). Estes modelos constituem uma extensão aos modelos baseados em MEF, como forma de ultrapassar a falta de flexibilidade desses modelos. O *frame* indica que informação deve ser fornecida pelo utilizador. Cada *slot* possui "*prompts*" associados e uma prioridade que determina a ordem pela qual o sistema tenta obter a informação para cada campo, até que a quantidade de informação pretendida seja atingida. Esta abordagem consegue ultrapassar duas limitações importantes da abordagem MEF, que são: i) apenas aceitar uma parte da informação de cada vez e, ii) não permitir a iniciativa do diálogo por parte do utilizador. Um *form* pode ser utilizado para consulta a uma base de dados ou pode ser submetido

para processamento quando todos os *slots* estão preenchidos. Os estados do diálogo dependerão de quais os *slots* se encontram preenchidos em cada momento. Um modelo de diálogo de topo deve garantir que todos os *slots* são preenchidos. O VoiceXML [VoiceXML, 2004] constitui o arquétipo de uma abordagem baseada em *frames*. A Figura 7 apresenta quatro *slots* (De, Para, Data e Hora) e ilustra o seu preenchimento a partir de duas possíveis expressões de entrada.

Vocalização de entrada (<i>input</i>)	De	Para	Data	Hora
Quero uma viagem do Funchal para Lisboa	Funchal	Lisboa		
... viagem para Lisboa a 29 Julho às 10 e 30		Lisboa	29 Julho	10:30

Figura 7: Ilustração de um *frame/form*

(adaptado de [Cavedon, 2005])

2.6.3 Abordagens baseadas em Planos

Os primeiros modelos baseados em planos, surgidos no campo da Inteligência Artificial, pretenderam resolver os problemas derivados da falta de flexibilidade dos sistemas baseados em máquinas de estados. Este modelo deriva da constatação de que os humanos planeiam as suas acções para atingir determinados objectivos [Jurafsky & Martin, 2006]; [Trung, 2006]. Um SD que empregue este modelo deve ser capaz de inferir sobre os objectivos do utilizador, bem como de construir e activar planos para levar a cabo a tarefa que permita fornecer o serviço requerido pelo utilizador. Este modelo pode ser visto como uma rede de transições de estados na qual os estados são gerados dinamicamente e não estão limitados a um conjunto finito e limitado. A criação de planos requer uma análise do comportamento racional dos participantes, o que permite interacções mais flexíveis que as permitidas por sistemas MEF [Delgado & Araki, 2005]. Na prática, a complexidade associada à construção e utilização de planos, dependente de processos de inferência, torna estes sistemas complexos e muito difíceis de modelar. O projecto TRIPS [Ferguson & Allen, 1998] representa o arquétipo para este tipo de modelos.

2.6.4 Abordagens baseadas em Information State Update (ISU)

A abordagem ISU [Cooper & Larsson, 1998]; [Bohlin *et al.*, 1999b]; [Traum *et al.*, 1999]; [Traum & Larsson, 2000]; [Larsson & Cooper, 2000]; [Traum & Larsson, 2003]; [Larsson *et al.*, 2004]; [Jurafsky & Martin, 2006] foi inicialmente desenvolvida nos projectos SIRIDUS [Cooper, 2000] e TRINDI [Traum *et al.*, 1999].

A *Information State Update* integra várias características das abordagens MEF, *Frames* e das abordagens por Planos. Inclui aspectos de estado do diálogo bem como o potencial para incluir representações semânticas e noções de obrigações, crenças e planos, os quais não são facilmente modeláveis por uma rede finita de estados. A ISU caracteriza-se por:

- Um *Information State* (IS): corresponde de certa forma a um "estado mental", contextualizando e representando o estado do diálogo. Cada *Information State* concreto fará a representação (em estruturas de dados abstractas) das propriedades que forem relevantes e passíveis de modelação. Esta estrutura de dados abstracta será acedida pelos diferentes componentes do GD, sendo o seu conteúdo actualizado a cada passo do diálogo;
- Um *motor de interpretação* (*Interpretation Engine*): interpreta cada contribuição do interlocutor humano como *dialogue moves*⁴. Estes *dialogue moves* contribuem para o progresso e actualização da informação de estado do diálogo;
- Um *Dialogue Move Engine* (DME): interpreta os *dialogue moves* de entrada e recorre a um conjunto de regras, ditas de actualização e/ou de selecção, para decidir, de cada vez, qual o *dialogue move* de saída do sistema (ie, qual a acção seguinte do sistema) [Ljunglof, 2000]; [Bos *et al.*, 2003], e;
- *Regras de actualização*: permitem que o DME actualize o *Information State* à medida que os *dialogue moves* são recebidos ou gerados. A(s) regra(s) a aplicar num determinado estado (de entre um conjunto de regras aplicáveis) dependerão do estado e das condições associadas a essas regras, as quais dependem da teoria de diálogo suportada/implementada.

A abordagem ISU permite modelar a relação entre diferentes tipos de informação, tais como: vocalizações, informação contextual, eventos não verbais e cenas visuais. Este último é um aspecto essencial ao processamento multimodal, [Burke *et al.*, 2003].

⁴ Um *dialogue move* [Ginzburg, 1996]; [Traum *et al.*, 1999]; [Bohlin *et al.*, 1999b]; [Larsson *et al.*, 2004]; [Quarteroni & Manandhar, 2007] constitui um acto ou uma acção associado a uma vocalização, no contexto de um diálogo. Pressupõe um conteúdo informativo e uma intenção de comunicação de uma expressão (ou de parte de uma expressão) [Stein & Thiel, 1993]. No Anexo II fazemos a sua descrição detalhada no contexto do Midiki.

A Figura 8 apresenta uma estrutura para um IS numa versão do SD Godis [Bohlin *et al.*, 1999b]; [Larsson *et al.*, 2000]; [Larsson *et al.*, 2000b]; [Hähnle & Larsson, 2003], o arquétipo dos SD baseados em ISU.

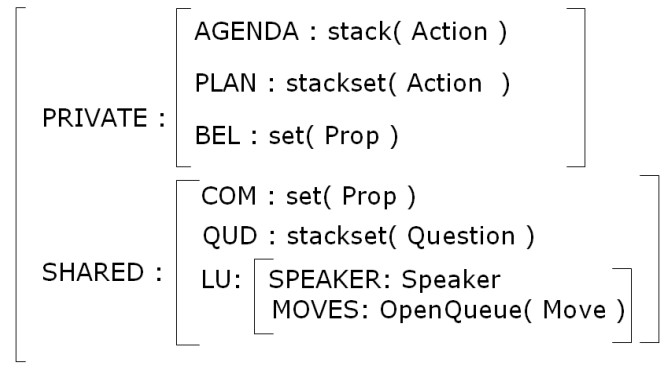


Figura 8: Estrutura de dados de um information state no Godis
(adaptado de [Larsson *et al.*, 2000])

A existência de uma parte privada e uma parte pública no IS significa que há informação partilhada com o utilizador e informação reservada do sistema.

Segundo Lemon (2008), em [Lemon, 2008], esta abordagem permite um nível de flexibilidade, adaptividade, robustez e naturalidade de interação, que é superior às abordagens MEF e *frames* (as quais dependem de representações em máquinas de estados finitos e modelam os diálogos basicamente como enormes grafos).

2.6.5 Abordagens baseadas em Agentes Colaborativos

Baseiam-se na perspectiva dos diálogos como um processo colaborativo entre agentes inteligentes. Ambos os agentes interlocutores trabalham conjuntamente com o fim de atingir um entendimento comum do diálogo [Ferguson & Allen, 1998]; [McTear, 2002]; [Trung, 2006]. Ao contrário das abordagens baseadas em estados (MEF, *frames*, e ISU), que se concentram na estrutura da tarefa, as abordagens colaborativas tentam capturar as motivações por detrás de um diálogo. As crenças de pelo menos dois participantes serão explicitamente modeladas. Um objectivo proposto, que será aceite pelo outro(s) participante(s), torna-se parte da crença partilhada e os parceiros trabalharão cooperativamente para atingir este objectivo.

As vantagens das abordagens colaborativas são a sua habilidade para lidar com diálogos mais complexos, envolvendo resolução colaborativa de problemas, negociação, etc. Em contrapartida apresentam uma complexidade superior às abordagens anteriores (MEF, *frames*, planos, e ISU) [Allen *et al.*, 2001].

2.6.6 Análise das abordagens

As cinco abordagens apresentadas constituem os modelos de referência para a modelação do núcleo de um gestor de diálogo, isto é, para a modelação do estado do diálogo, das condições que permitem alterar esse estado e da forma como se pretende/espera que o diálogo decorra. Todas estas abordagens possuem variantes e em vários sistemas criados podemos encontrar, em simultâneo, características tipicamente associadas a diferentes abordagens [Trung, 2006].

A Tabela 1, adaptada de [Allen *et al.*, 2001], compara as abordagens anteriores através de três aspectos: tipo de tarefa (típica), complexidade relativa das tarefas suportadas e fenómenos de diálogos suportados:

Tabela 1: Comparação das diferentes abordagens à gestão do diálogo

Técnica utilizada na abordagem	Tipo de tarefa	Complexidade (crescente) da implementação	Fenómenos de diálogo tratados
Máquina de Estados Finitos	Ex. Pedido de informação numa lista telefónica ou de informação bancária.	A abordagem mais simples. É inflexível e pouco natural (apenas um estado, pré-definido, resulta de uma transição).	Utilizador responde a questões colocadas pelo sistema e a iniciativa parte sempre do sistema.
Baseado em <i>slots</i> / frames	Ex. Efectuar uma reserva de viagem. Apenas uma parte da informação (ie, um <i>slot</i>) pode ser indicado de cada vez. A ordem de preenchimento dos <i>slots</i> não é rígida.	Mais complexa e mais flexível que a anterior, mas não é adequado para interacção em linguagem natural. Comum em sistema comerciais (VoiceXML).	O utilizador pode colocar questões e receber clarificações simples por parte do sistema. Permite alternância da iniciativa no diálogo.
Information State Update	Ex. Acesso multimodal a sistemas de reservas de bilhetes para o cinema (ou outro tipo de reservas semelhantes).	Complexidade média. Permite linguagem natural e multimodalidade.	Alternância entre tópicos pré determinados (dentro do modelo de domínio). Permite iniciativa mista. Diferentes teorias de diálogo podem ser modeladas num sistema ISU.
Modelos	Ex. Consultor de	Complexidade	Estruturas de tópico

baseados em planos	concepção de uma cozinha ou de uma rede local (LAN).	elevada. Há necessidade de um mecanismo de inferência, o que torna a sua implementação mais complexa que as anteriores.	geradas dinamicamente (objectivos e intenções são “modelados” dinamicamente).
Modelos baseados em agentes colaborativos	Ex. Gestão de situações de desastre/emergência	A mais complexa de todas as abordagens. Cada agente autónomo actuará como um sistema autónomo.	Negociação colaborativa.

Qual a abordagem mais adequada para uma determinada aplicação?

De acordo com Allen, em [Allen *et al.*, 2001], da abordagem MEF para a abordagem colaborativa há uma complexidade crescente, ao nível do modelo de diálogo, modelo de tarefa e domínios de aplicação. Agentes conversacionais (colaborativos) que incorporem princípios de racionalidade e cooperação podem parecer a solução óbvia uma vez que são os que mais se aproximam das competências humanas em termos de conversação e capacidade de diálogo. Isso é verdade para aplicações destinadas à resolução de problemas, com soluções negociadas, para as quais as outras abordagens não são suficientes. Por outro lado, segundo Robinson, em [Robinson *et al.*, 2004], para aplicações menos complexas, com tarefas restritas (o que acontece na maioria dos casos práticos pretendidos a nível comercial), as abordagens mais apropriadas resumem-me muitas vezes às técnicas mais básicas, tais como *frames*. Entre esses dois níveis de complexidade encontram-se os sistemas do tipo ISU. A maioria dos SD comerciais existentes baseia-se em abordagens do tipo frames/VoiceXML, os quais permitem modelos de diálogo mais robustos. Já em sistemas em que a utilização de multimodalidade constitua um requisito, soluções baseadas em ISU permitirão implementações práticas viáveis.

2.7 Ferramentas para a modelação de diálogos

Uma implementação completa, de raiz, de um GD para o suporte a determinada tarefa num determinado domínio exige conhecimentos de áreas tais como linguística, conhecimentos específicos do domínio de aplicação, configuração de bases de dados, integração de componentes, etc. Congregar todas essas fontes apenas através de linguagens de programação e *scripting* tradicionais não é uma tarefa fácil e, com

excepção de soluções comerciais usualmente dispendiosas e em geral baseadas em VoiceXML, não existem metodologias claras e/ou standardizadas para o suporte à autoria de diálogos em GDs. Um dos esforços na disponibilização de tais ferramentas está relacionado com o trabalho de Staffan Larsson e da sua equipa na Universidade de Gotemburgo, onde foi criado o Trindikit [Traum *et al.*, 1999]. Infelizmente, existe falta de ferramentas de autoria de diálogos que sejam acessíveis, quer em termos de custos e abertura do código, quer em termos de facilidade da modelação de diálogos [Quintal & Sampaio, 2007].

Existem poucas ferramentas disponibilizadas livremente para este fim. Selecionamos três soluções, representando as três abordagens mais comuns e mais utilizadas na prática (MEF, *Frames* e ISU, e deixando de fora as abordagens baseadas em planos e em agentes colaborativos): o CSLU Toolkit [McTear, 1998]; o Midiki [Burke *et al.*, 2003]; [Burke, 2005b] e o Loquendo SDS Studio [Romellini, 2003].

O CSLU Toolkit suporta uma abordagem baseada em MEF, o Midiki é representativo da abordagem ISU e ambos são disponibilizados gratuitamente, sendo que o Midiki está totalmente implementado em Java e disponível em *Open Source* e o CSLU Toolkit está implementado em “C” (com uma licença mais restritiva que a do Midiki). O Loquendo SDS Studio é uma solução comercial e proprietária, representativa das soluções baseadas em Frames/VXML. Passamos a apresentar brevemente essas três ferramentas/soluções.

2.7.1 O CSLU Toolkit

Trata-se de uma ferramenta RAD (Rapid Application Development), que permite a prototipagem de sistemas de diálogo baseados em MEF. Foi desenvolvida pelo *Center for Spoken Language Understanding*⁵ [McTear, 1998]. Suporta nativamente o reconhecimento de fala em Português e integra o Festival para síntese de fala [Black *et al.*, 2008]. Está disponível apenas para máquinas com sistema operativo “*Microsoft Windows*”TM. A Figura 9 ilustra as interfaces desta ferramenta:

⁵ <http://cslu.cse.ogi.edu/>

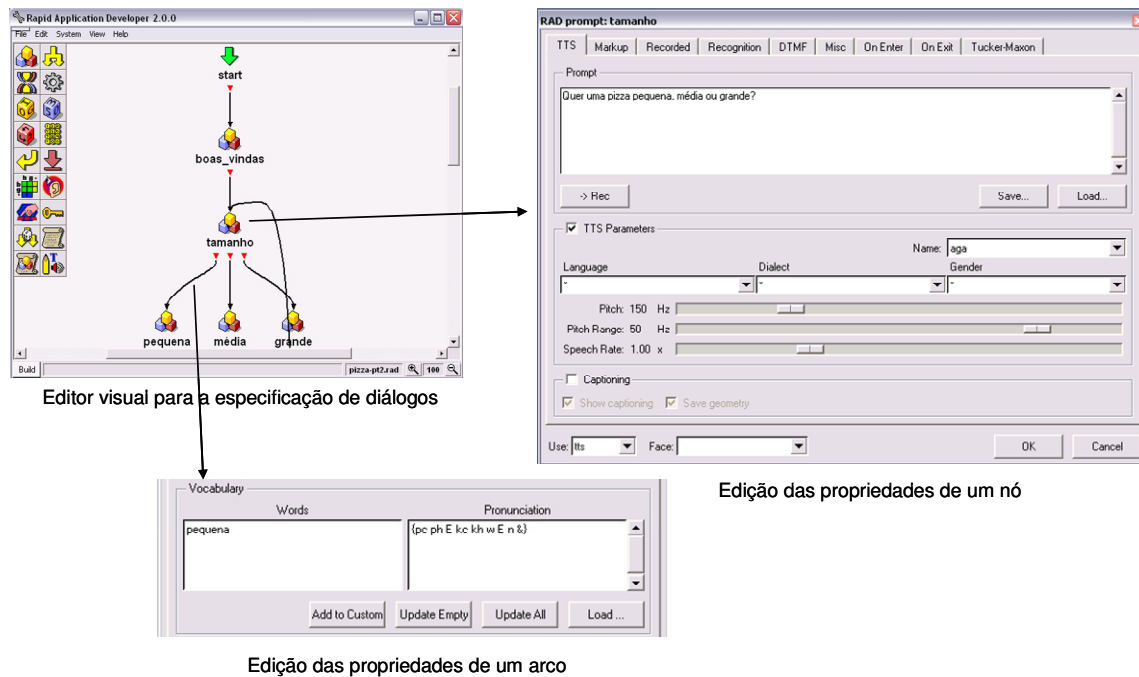



Figura 9: Interfaces de trabalho do CSLU Toolkit

A criação de protótipos com o CSLU Toolkit é bastante acessível, através de um ambiente gráfico/visual (que testámos). Os diálogos são construídos pela inclusão e parametrização de objectos. Um estado é representado por um ícone  e é sobre este objecto que são criados os *prompts* (mensagens/questões) do sistema dirigidos ao utilizador. Suporta acções para eventos “On Exit” e “On Enter”, entre outros. Os diferentes “estados” são ligados por transições que correspondem às possíveis respostas do utilizador. Existem ainda objectos do tipo subdiálogo, condições e execução de acções “externas” (através de linguagem de *scripting TCL* [Ousterhout, 1988]). Além da interacção por fala, também suporta interacção por telefone através de teclado (tons) e a interacção por janelas do tipo WIMP. Um diálogo pode ser executado após compilação e o sistema pode ser acedido remotamente, como um *server/daemon*.

2.7.2 O Loquendo Spoken Dialogue System

O *Loquendo Studio* [Romellini, 2003]; [Danieli & Manca, 2005]; [Loquendo, 2005] é composto por um conjunto abrangente de ferramentas/módulos para a criação de serviços, testes, depuração e simulação de sistemas de diálogo baseados em *Frames/VoiceXML*. Funciona em ambiente *web/browser* e pode ser configurado tanto em modo de utilizador (programador) individual, quer em ambiente de “fábrica de software”. Suporta vários idiomas, incluindo o português. A ferramenta inclui seis módulos funcionais:

- **Módulo de configuração:** diz respeito à configuração de parâmetros de serviço, ie, parâmetros a serem fornecidos ao interpretador VoiceXML (ex. gramáticas, ficheiros áudio, idioma padrão, etc);
- **Depurador VoiceXML:** Permite a depuração de código VoiceXML;
- **Módulo gramatical:** Permite realizar testes sintáticos e semânticos e a compilação de gramáticas;
- **Módulo de simulação:** Permite simular a navegação num serviço de voz através de um computador equipado com uma placa de áudio, sem exigir um telefone ou a plataforma final, e;
- **Módulo de exportação:** Para a criação de pacotes a incluir nas plataformas de voz (incluindo arquivos de áudio).

Na modelação de diálogos destaca-se a possibilidade de poderem ser aplicadas várias estratégias de recuperação de erros: ex. Prevenção de repetição de erros; Alternância automática entre estilos de diálogo em situações extremas (ex.: sucessivos *timeouts*); Verificação de consistência entre respostas relacionadas; etc. A relação entre as diferentes partes de um sistema desenvolvido e disponibilizado através do *Loquendo Studio* é apresentada na Figura 10. O diagrama ilustra a integração entre o Gestor de Diálogo (*Loquendo SDS*), a ferramenta de autoria (o *Loquendo SDS Studio*) e as plataformas VoiceXML disponibilizadas ao utilizador final. Não testámos esta ferramenta.

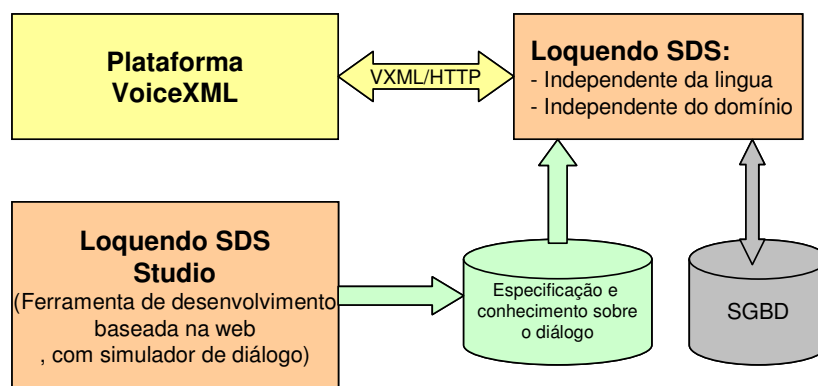


Figura 10: Componentes de um sistema desenvolvido com base no Loquendo SDS

2.7.3 O Midiki

O Midiki [Burke *et al.*, 2003]; [Burke, 2005b] permite a prototipagem de sistemas de diálogo baseados em ISU. É apresentado em detalhe nos capítulos 4 (O gestor de diálogo Midiki), 5 (Implementação da Ferramenta de Autoria) e 6 (Caso de Estudo), pelo que fazemos aqui apenas uma introdução a essa ferramenta. Na versão base do

Midiki os diálogos são criados trabalhando directamente sobre classes Java. Cada diálogo implementado é composto por componentes do tipo “Domínio” (contém planos e atributos), “Léxico” (inclui entradas lexicais/gramaticais para o diálogo), “Cells” (define estruturas de dados de suporte) e “Handlers” (implementa o acesso a serviços externos ao gestor de diálogo). O Midiki disponibiliza módulos de entrada e de saída básicos e algumas facilidades de *debugging*. Podem ser concebidos e integrados novos módulos e funcionalidades. Este GD pode ser integrado em ambientes distribuídos, via *Galaxy* [Seneff *et al.*, 1998]; [MITRE, 2003] ou *OAA* [Martin *et al.*, 1999]. Na versão que utilizamos apenas são permitidos diálogos “verbais” (texto escrito ou texto proveniente do reconhecimento de fala), em linguagem natural. No Midiki, um diálogo pode ser executado em ambiente Java após compilação. A Figura 11 ilustra a edição sobre uma classe Java na versão base do Midiki (à esquerda) e a edição recorrendo à ferramenta criada no âmbito deste trabalho (à direita):

```

package org.mitre.dm.qud.domain.ATMDomain;

import java.util.*;
import java.util.logging.*;

import org.mitre.midiki.logic.*;
import org.mitre.midiki.state.*;
import org.mitre.dm.qud.domain.*;

public class ATMDomain extends DomainCell {

    private static Logger logger =
        Logger.getLogger("org.mitre.dm.qud.domain.ATMDomain");

    public ATMDomain() {
        super();
    }

    protected void initializeAttributes() {
        ArrayList atrib30 = new ArrayList();
        atrib30.add(("ajuda"));
        atrib30.add(("opcoes"));
        atrib30.add(("adeus"));
        atrib30.add(("ola"));
        atrib30.add(("perdao"));
        atrib30.add(("pretendo"));
        attributeMap.put("generalAttribut

```

Attributes

BACK PRINT DELETE

Attribute type:

Description:

Attribute values: [click here](#)

Save

Attributes and synonyms (step 2 of the methodology): current project is /

NEW ATTRIBUTE VALUE PRINT

Value	Synonyms	Attribut
multibanco	click here	metodosPagam
dinheiro	click here	metodosPagam
cheque	click here	metodosPagam
credito	click here	metodosPagam
transferencia	click here	metodosPagam

Figura 11: Exemplo de classe Java do "domínio" de um diálogo e de um ecrã de edição na ferramenta web

2.7.4 Análise das ferramentas

As três ferramentas apresentadas constituem uma amostra do tipo de recursos disponíveis para a modelação/autoria de diálogos recorrendo a cada uma das abordagens associadas a essas ferramentas (MEF, *Frames* e ISU). A Tabela 2, adaptada de [McTear, 2002], apresenta uma comparação destas três

abordagens/ferramentas segundo quatro critérios: tipos de entrada permitidos, tipos de verificação das entradas, modelo de diálogo e modelo de utilizador.

Tabela 2: Comparação das três ferramentas (CSLU Toolkit, Loquendo SDS e Midiki) e respectivas abordagens para gestão do diálogo

	MEF: CSLU Toolkit	Frames/VoiceXML: Loquendo SDS	ISU: Midiki
Tipo de entrada permitido	Palavras ou orações isoladas	Frases simples, em LN. Recebe apenas uma parte da informação de cada vez	Linguagem natural. Pode suportar entradas multimodais
Tipo de verificação da entrada	Confirmação explícita, de cada entrada	Confirmação explícita e/ou implícita	Permite estratégias de entendimento (<i>grounding</i>), com confirmação explícita ou implícita
Modelo de diálogo	Controlo do diálogo representado explicitamente pelo diagrama de estados (define implicitamente o estado do diálogo)	Representação explícita dos estado através de <i>slots</i> . Controlo do diálogo representado por um algoritmo de controlo	Modelo de intenções / obrigações do sistema (planos); crenças. Representação do histórico e do contexto do diálogo. Pode ser multimodal
Modelo de utilizador	Modelo simples das características e preferências do utilizador	Modelo simples das características e preferências do utilizador	Modelo de intenções e de obrigações do utilizador; objectivos e crenças

Para além dos três gestores de diálogo apresentados anteriormente (e respectivas ferramentas de suporte), vários outros gestores de diálogo e ferramentas são frequentemente referidos na literatura, dos quais seleccionamos os seguintes:

- **TRINDIKIT** [Traum *et al.*, 1999]: *Toolkit* para a prototipagem de sistemas de diálogo ISU. Compatível com várias soluções terceiras de reconhecimento e geração de fala. O GoDis é um sistema de diálogo que demonstra várias aplicações do TrindiKit;

- **ARIADNE** [Denecke, 2000]: *Toolkit* para a prototipagem rápida de sistemas de diálogo para interacção por fala, desenvolvido pelo *Interactive Systems Lab* da Universidade de Carnegie Mellon e pela Universidade de Karlsruhe;
- **BEVOCAL CAFÉ** [Bevocal, 2002]: Ambiente de desenvolvimento, completo e gratuito, para sistemas baseados em VoiceXML. Disponibilizado pela Nuance Communications;
- **DIPPER** [Bos *et al.*, 2003]: *Toolkit* para a prototipagem de sistemas de diálogo, com base em ISU e utilizando o protocolo de comunicação OAA. Desenvolvido pelo *Edinburgh Language Technology Group*, e;
- **Rubin** [Buckley & Benz Müller, 2005]: *Toolkit* para a prototipagem de sistemas de diálogo, com base em ISU. Desenvolvido pela CLT Sprachtechnologie.

2.8 Um projecto de referência

O projecto Europeu TALK⁶ [Georgila & Lemon, 2004], decorreu entre 2004 e 2007, e teve por objectivo o desenvolvimento de novas tecnologias para sistemas de diálogo humano-computador multimodais, adaptativos (sistemas que aprendam a partir da interacção com os utilizadores), mais robustos e mais intuitivos que os existentes, para diferentes domínios de aplicação. Incluímos este projecto como uma referência pelos seguintes motivos: i) prosseguiu o desenvolvimento da ISU, que actualmente é a abordagem que apresenta maior interesse prático na criação de novos sistemas de diálogo multimodais; ii) No âmbito do TALK foram produzidos resultados concretos, tendo envolvido parceiros académicos e industriais de elevado crédito, com influência no estado da arte da investigação e no lançamento de novos produtos no mercado.

O TALK prosseguiu o trabalho realizado no projecto TRINDI (no âmbito do qual foi desenvolvido o ISU) e veio propor o *Extended ISU* [Amores *et al.*, 2006]. O *Extended ISU* dá suporte a interacções combinando informação verbal e gráfica, onde o que pode ser exibido inclui texto (ex. tabelas ou listas) ou imagens (ex. mapas). Isso implica, em particular, uma representação explícita das modalidades disponíveis, da sua capacidade para veicular conteúdo/informação, e a manutenção de controlo sobre que informação foi veiculada por que modalidade.

Este projecto juntou parceiros académicos e industriais, com vista a obter resultados que possam ser directamente integrados nas unidades de I&D dos parceiros industriais. Para demonstrar estas novas tecnologias, desenvolvidas no âmbito do TALK, foram construídos sistemas protótipo em veículos e em habitações. Alguns

⁶ Talk and Look - Tools for ambient linguistic knowledge

destes resultados estão a ser aplicados industrialmente, como é o caso do reprodutor de músicas *BMW iDrive* [Lemon, 2008].

2.9 Conclusões

A maior parte dos sistemas de diálogo com suporte à interacção por fala (e/ou outras modalidades) ainda exigem demasiadas acções conscientes da parte do interlocutor humano. Além disso, a probabilidade de falhas de reconhecimento continua a ser, em geral, elevada. Um dos desafios actuais é as máquinas irem assumindo uma responsabilidade crescente na manutenção da integridade dos canais, sendo capazes de o fazer de forma robusta, detectando e interpretando correctamente os diversos *inputs* do utilizador e eventuais alterações de contexto [Cavedon, 2005]; [Melichar, 2008]. Relativamente às modalidades que devem ser utilizadas em cada momento é ainda, em geral, uma escolha do utilizador e a melhor forma das máquinas poderem antecipar e propor determinada(s) modalidade(s) será recorrendo a algoritmos de aprendizagem. De uma forma genérica, o grande desafio actualmente reside na criação de sistemas multimodais mais robustos, que aprendam e se adaptem.

Cavedon, em [Cavedon, 2005] aponta como principais temas de pesquisa e desafios actuais, os seguintes:

- **Aperfeiçoamento na adaptação:** Os sistemas devem ser capazes de adaptação dinâmica, de diferentes formas: adaptação aos ambientes em que são utilizados (modalidade); adaptação às necessidades e preferência do utilizador (personalização), e; adaptação às mudanças nas tarefas e no contexto. Implica a utilização de estratégias de diálogo adaptivas (aprendizagem a partir das interacções com os utilizadores) e a necessidade de adaptar a geração de linguagem natural ao utilizador;
- **Incorporação de novas modalidades nas plataformas** (ex.: olhar e gestos);
- **Gestão de diálogo múltiplo** (ex. gestão de reuniões com vários participantes humanos: fala e gestos);
- **Criação de agentes conversacionais** com forma e comportamento “humanos” (ex. apresentação de expressões faciais e indicação de emoções);
- **Proposta de arquitecturas e sistemas mais robustos:** combinação de diferentes fontes de informação (ex. multi-domínio; utilização de recursos extensos: ex. léxico do WordNet [Miller, 2006] e de outras fontes de referência disponíveis). Detecção e reparação de erros melhorada;
- **Standardização:** há necessidade de definição de *standards* que venham permitir a reutilização e melhorar a usabilidade de sistemas de diálogo, e;

- **Utilização de ambientes distribuídos:** necessidade dos sistemas suportarem melhor a distribuição e a concorrência (sistemas ubíquos), nomeadamente na utilização de terminais leves, tais como PDAs (o que também requer aperfeiçoamentos de *hardware*).

Capítulo 3

O Gestor de Diálogo

Midiki

Este capítulo apresenta o gestor de diálogo Midiki e está estruturado da seguinte forma: Na secção 3.1 justificamos porque seleccionamos o Midiki; Na secção 3.2 é apresentado como funciona a gestão do diálogo neste GD; Seguidamente, na secção 3.3, apresentamos a sua arquitectura e o information state (IS); A secção 3.4 apresenta uma conclusão a este capítulo.

O Midiki⁷ [Burke *et al.*, 2003]; [Burke, 2005b]; [Quintal & Sampaio, 2007] foi desenvolvido por Carl Burke (Phd) e a sua equipa na MITRE Corporation⁸ e está integralmente disponível em regime *Open Source*, na *SourceForge*⁹. O Midiki recorre ao *Information State Update* como abordagem para a gestão do diálogo.

⁷ MITRE Dialogue Kit

⁸ <http://www.mitre.org/>

⁹ <http://midiki.sourceforge.net>

3.1 Porque seleccionamos o Midiki

Conforme descrito por Carl Burke em [Burke, 2005], o Midiki foi construído com o objectivo de permitir implementar gestores de diálogo relativamente sofisticados, minimizando tanto quanto possível o esforço de desenvolvimento e de portabilidade. Um dos objectivos era que pudesse adaptar-se de forma relativamente fácil à interacção multimodal. Outro objectivo dos autores do Midiki foi colmatar a falta de equilíbrio dos GD existentes no que respeita aos modelos linguísticos face aos aspectos de implementação prática, ou seja, desenvolver um GD que obedecesse aos requisitos dos linguistas, suportasse comportamentos sofisticados e, ao mesmo tempo, oferecesse visibilidade total sobre as suas operações internas. Devia também ser agnóstico relativamente a sistemas operativos e teorias de diálogo.

Os motivos para a nossa escolha do Midiki como gestor de diálogo para este trabalho foram os seguintes:

- Baseia-se em representações *Information State Update* (ISU) que são, por um lado, mais flexíveis que os sistemas tradicionais baseados em MEF, e por outro lado viáveis do ponto de vista prático (suportando multimodalidade e linguagem natural);
- Está disponível em *Open Source*, tendo sido totalmente implementado em Java (o que nos dá a liberdade para adaptações ou alterações de qualquer parte do código). Além disso, o facto de querermos disponibilizar posteriormente, em regime de código aberto, de forma acessível, todo o código fonte criado contribui para a escolha de tecnologias disponíveis em código aberto;
- Estando totalmente implementado em Java pode ser compilado e executado em qualquer sistema operativo de referência;
- Permite a implementação de diferentes teorias de diálogo;
- Pode suportar interacção multimodal. Embora isso implique a necessidade de adaptar o *Information State*, o Midiki foi implementado com o intuito de poder suportar interacção multimodal;
- Permite integração com as duas principais *frameworks* disponíveis livremente para a criação de SD em ambientes distribuídos: a *Open Agent Architecture* e a *Galaxy Communicator*;
- É altamente modular e permite adaptar facilmente uma funcionalidade ou classe às necessidades do investigador;

- Baseou-se numa das versões anteriores do Trindikit, melhorando-a nalguns aspectos. Ao passo que o Trindikit requer um interpretador Prolog¹⁰ comercial (e proprietário), o Midiki não depende de *software* de terceiros, e;
- Finalmente, apesar destas propriedades muito interessantes, o Midiki não dispunha ainda de qualquer ferramenta de suporte à autoria de diálogos, o que nos deu a oportunidade de criá-la.

3.2 Estrutura do diálogo

Como já foi referido, uma limitação do Midiki era não existir ainda nenhuma ferramenta de autoria disponível para este GD (a única forma de criar diálogos era trabalhando directamente sobre classes Java). A metodologia e a ferramenta de autoria que propomos permitem a autoria de diálogos de forma mais intuitiva e mais acessível, facilitando e acelerando a prototipagem de sistemas de diálogo com base no Midiki.

3.2.1 Elementos que compõem o diálogo

De uma forma geral, e no Midiki em particular, um diálogo típico será composto por:

- Uma abertura do diálogo (um início ou estabelecimento de uma conversação);
- Pela realização da conversação (implica normalmente uma alternância de perguntas e respostas por ambas as partes):
 - Identificação do tópico e tarefa a tratar, ie, identificar o que é pretendido pelo utente do sistema/serviço;
 - Identificação da acção a executar no âmbito do tópico identificado (este passo poderá ser repetido durante o diálogo), e;
 - Execução da tarefa seguindo um determinado plano que permita atingir o seu objectivo (a tarefa identificada aquando da identificação do tópico).
- Por uma conclusão do diálogo, com a obtenção do resultado pretendido por parte do utilizador (o objectivo do diálogo).

No contexto do Midiki, aos elementos de conversação que podem ser colocados num plano e que permitem levar a cabo um diálogo chamamos estratégias. Cada diálogo é suportado por um plano, o qual é composto por estratégias.

O diálogo estrutura-se assim através de planos, os quais pertencerão a um domínio (existe no Midiki uma componente formal “domínio” na modelação de um diálogo) e

¹⁰ <http://www.sics.se/sicstus/>

serão compostos pelo encadeamento de estratégias. Cada estratégia pode desencadear um ou mais *dialogue moves* de saída. Por sua vez as contribuições do utilizador são passadas ao sistema na forma de *dialogue moves* de entrada. Nas secções seguintes apresentamos em pormenor cada um dos elementos que estruturam um diálogo no Midiki.

3.2.2 O domínio

No Midiki, o domínio de um diálogo é um recurso que descreve entidades: planos, estratégias (as quais despoletam *moves*) e atributos, com o objectivo de realizar determinada tarefa. Planos e estratégias constituem os elementos base da modelação do diálogo.

3.2.3 Planos

Um plano é uma colecção estruturada de acções/estratégias concebida para levar a cabo uma tarefa. Um plano constitui a estrutura dorsal de um diálogo. Os planos descrevem e definem interacções típicas entre o utilizador e o sistema (de diálogo), as quais são concretizadas através da execução das estratégias do plano.

A execução de um plano implica identificar tópicos, obter e facultar informação relevante no âmbito dos tópicos identificados e executar as acções previstas. Num diálogo no qual possam existir vários planos alternativos (ex.: um sistema que executar várias tarefas alternativas independentes), será necessário executar inicialmente um plano de topo que permita determinar qual dos planos alternativos deve ser executado. O plano de topo consiste normalmente em informar o utilizador de quais os serviços disponíveis, perguntar-lhe o que pretende fazer e, de acordo com a sua resposta identificar o tópico e direccionar a acção para um dos planos alternativos principais. Visto como um subplano, um plano pode ser invocado por verificação de uma condição. Finalizada a execução do subplano, é devolvida ao plano principal a condução da continuação do diálogo.

3.2.4 Dialogue moves

Os *dialogue moves* são parte integrante da abordagem ISU (ver secção 2.6.4) e, por consequência, são parte integrante do funcionamento do Midiki. Num *input* do utilizador para o sistema, cada *dialogue move* é extraído de uma vocalização ou expressão. As vocalizações são parte de cada alternância do diálogo (no sentido de que cada interlocutor fala “na sua vez”), as quais correspondem a unidades do diálogo. Por sua vez, cada estratégia, de um plano, executada pelo sistema faz desencadear

dialogue moves de saída. Uma descrição detalhada dos *dialogue moves* suportados no Midiki é feita no Anexo II.

3.2.5 Estratégias

O elemento estratégia encontra-se um nível abaixo do elemento plano e um nível acima do elemento *dialogue move*. As estratégias podem ser conversacionais ou não conversacionais. Uma estratégia conversacional deve permitir dialogar com o utilizador humano:

- Colocando-lhe **questões** com vista a obter determinada informação. Esta acção corresponderá a uma estratégia ***findout***: implica colocar uma questão e obter informação do utilizador, de forma a fazer corresponder a informação que será dada pelo utilizador com a informação pretendida pelo sistema;
- **Informar** o utilizador, o que corresponderá a uma estratégia ***inform***: informar o utilizador de um resultado de uma query ou apresentar algum output relacionado com regras de conversação / sociais, e;
- Estabelecer **proposições** que irão definir o contexto do diálogo (o seu conteúdo no decorrer da “conversa”) e que passam a pertencer ao conhecimento partilhado entre os interlocutores (ex. uma estratégia ***assert***).

A execução de uma *estratégia* desencadeia normalmente um ou mais *dialogue moves*. Uma descrição detalhada das estratégias suportadas no Midiki é feita no Anexo II.

Além da especificação da componente de domínio, a modelação de um diálogo no Midiki fica completa com a inclusão das componentes léxico e *cells*, as quais disponibilizam conteúdo lexical e estruturas de dados de suporte ao diálogo.

Nesta secção expusemos os elementos que compõem a estrutura de um diálogo e permitem modelar/especificar diálogos no Midiki. Na secção seguinte apresentamos a arquitectura deste GD, nomeadamente os agentes que compõem o Midiki.

3.3 Arquitectura

O Midiki está estruturado numa arquitectura baseada em agentes, em que cada agente desempenha cada uma das funções principais do GD: interpretar entradas; gerir informação de domínio; decidir o que fazer a seguir (e actualizar o estado) e gerar uma saída. Não existe um controlo global no Midiki, no sentido em que um algoritmo implementado de forma procedimental chama, alternativamente, cada agente (*InterpretAgent*, *DME Agent* e *GenerateAgent*). No Midiki, cada agente pode registar um *listener* no agente *ISControl* para que seja notificado sempre que um

atributo é modificado. O agente *ISControl* serve o propósito de controlar o acesso ao *Information State* (a estrutura de dados abstracta) [Burke, 2005]. O agente *Input/Output* disponibilizado não cumpre uma função nuclear do GD e pode ser adaptado consoante as necessidades de *entrada/saída* de cada sistema criado. O diagrama da Figura 12 ilustra esta arquitectura por agentes:

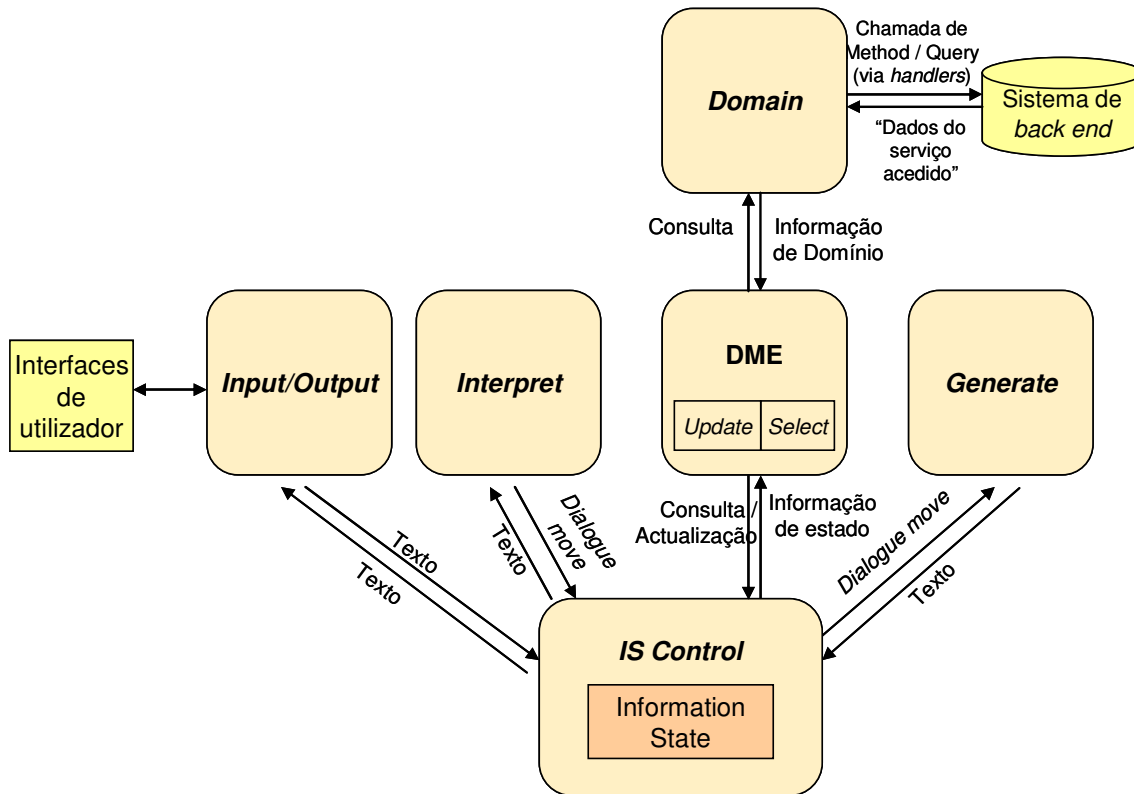


Figura 12: Arquitectura (por agentes) do Midiki

Este diagrama ilustra o que se passa no caso em que a entrada e saída no agente *Input/Output* é do tipo unimodal (ie, suporte apenas a texto). Num sistema multimodal teríamos informação multimodal nas trocas de dados/informação entre os agentes e no seu processamento interno (onde no diagrama da Figura 12 se lê “texto” teríamos outro tipo de codificação). Na Tabela 3, adaptada de [Burke, 2005], descrevemos os agentes que compõem esta arquitectura, considerando as trocas de dados tal como indicadas no diagrama da Figura 12.

Tabela 3: Os diferentes agentes que compõem o Midiki

Agente	Breve descrição
<i>Input/Output</i>	Disponibiliza uma interface de entrada e saída (na versão base através de uma janela para entrada e saída de texto). É o componente responsável pela interligação com as interfaces de utilizador. No caso da interacção

	“simples”, por texto, a entrada e saída consiste de texto não semantizado (texto não processado).
<i>Interpret</i>	É o motor de interpretação do sistema. Recebe “texto” em linguagem natural (produzido pelo utilizador) como <i>input</i> e determina a sua semântica, ie, realiza o <i>parsing</i> dessa entrada para identificar os <i>dialogue moves</i> de entrada. Divide o input em tokens e analisa-o à procura de <i>dialogue moves</i> .
<i>DME</i>	O <i>Dialogue Move Engine</i> é o componente central do Gestor de Diálogo. Ao receber um <i>dialogue move</i> (de entrada ou de saída), executa/aplica as regras de actualização (<i>update</i>) e selecciona a próxima acção de sistema, através de uma regra de selecção (<i>select</i>). A cada passo o <i>information state</i> é actualizado pelos “efeitos” da aplicação/execução das regras.
<i>Generate</i>	É o motor de geração do sistema. Selecciona o texto de <i>output</i> (a partir do léxico) reflectindo o <i>move</i> de saída gerado pelo sistema (pelo DME). Estes <i>dialogue moves</i> resultam em geral da execução das estratégias de um plano.
<i>ISControl</i>	Controla o acesso ao <i>Information State</i> , actualizando os respectivos campos/atributos. Suporta o disparo adequado de “data listeners” para os agentes que os tenham subscrito.
<i>Domain</i>	Disponibiliza uma estrutura (ex. informação dos planos) e raciocínio (ex. determinar a relevância, a veracidade ou a precisão de um <i>input</i> do utilizador) para o domínio. Compete a este agente determinar a relevância de <i>inputs</i> do utilizador relativamente aos planos/estratégias e obter novos planos quando necessários.

3.3.1 Estrutura modular

Todo o sistema do Midiki está implementado em Java e é composto por módulos. Na estrutura de pastas do Midiki (Figura 57), é na pasta de nome “dm” (*dialogue manager*) que se encontram os módulos relativos às teorias de diálogo e aos diálogos propriamente ditos que sejam construídos com o Midiki. Na pasta “dm -> qud” encontramos os componentes que implementam a teoria de diálogo “Questions Under Discussion” (QUD) [Ginzburg, 1996]; [Traum *et al.*, 1999]. É ao nível da “qud” que são implementadas algumas das classes relacionadas com a infra-estrutura do diálogo (ex.: *InterpretAgent*, *GenerateAgent* e *IOAgent*). É também a este nível que se encontram algumas das classes que implementam a ISU, tais como *ISControlAgent* ou *DMEAgent*, por exemplo. Esta estrutura modular do Midiki define que partes dependem de outras partes e qual o seu grau de independência e estes agentes terão

uma implementação específica consoante a teoria de diálogo que se pretenda empregar. Sob uma pasta *domain* (ver Figura 56) incluiremos os modelos de diálogos concretos que permitirão ao gestor de diálogo levar a cabo determinada tarefa. As representações da estrutura do *Information State* (IS) podem diferir de sistema para sistema, dependendo da teoria de diálogo adoptada e de outros requisitos (por exemplo, consoante as modalidades a suportar). Na secção seguinte descrevemos como é constituído um IS no Midiki.

3.4 O Information State (IS)

Nesta secção utilizamos a designação *Information State* no seu sentido mais restrito, relativo à estrutura de dados subjacente ao IS. Para a acepção mais geral temos utilizado normalmente a designação ISU (*Information State Update*), considerado como o componente do sistema que permite a representação, inspecção e actualização do estado do diálogo (incluindo o IS, o motor de interpretação, o DME e as regras de actualização). A estrutura de dados abstracta *Information State* é constituída por campos estáticos e dinâmicos e na sua definição é composta por registos, pilhas, filas, conjuntos e outros tipos de dados mais básicos. Os componentes estáticos não mudam durante o diálogo (por ex. conhecimento específico do domínio ou da teoria de diálogo). Os componentes dinâmicos dividem-se em partes privada e partilhada. A informação privada inclui, por ex., as "crenças" do próprio sistema, com base nas informações recebidas do utilizador e das acções que o sistema executa no decorrer do diálogo. A informação partilhada consiste em conhecimento mútuo (por ex. o último *dialogue move* gerado).

3.5 Suporte a diferentes teorias de diálogo

A teoria de diálogo é utilizada como estratégia global de conversação, de forma independente de um diálogo ou domínio em particular. No caso do Midiki, a teoria de diálogo implementada por omissão é a "Questions Under Discussion" (QUD) [Ginzburg, 1996]. O Midiki pode suportar outras teorias de diálogo, se forem implementadas, tendo em conta que a consideração de uma determinada teoria de diálogo implica a modificação do módulo que a implementa, e consequentemente dos módulos e/ou classes dependentes desse módulo.

3.5.1 A teoria de diálogo *Questions Under Discussion* (QUD)

"Questions Under Discussion" pode ser traduzido para Português como "Questões em Discussão". Esta teoria de diálogo constitui uma forma de explicar e formalizar a estrutura dos diálogos [Traum *et al.*, 1999] e foi inicialmente proposta por Jonathan

Ginzburg em [Ginzburg, 1996], com base na definição de uma vista estruturada do contexto de um diálogo. Uma QUD é implementada através das regras no DME (regras de actualização e de selecção) e de estruturas de dados apropriadas no IS. No sentido restrito da estrutura de dados do IS, o tipo/campo QUD é uma pilha contendo questões à “espera” de serem esclarecidas, quer as questões provenientes de um plano do diálogo, quer as questões que tenham sido introduzidas no diálogo pelo utilizador. Uma questão permanecerá na QUD até que seja esclarecida. À medida que o sistema recebe novos *inputs* do utilizador, vai verificando se esses *inputs* resolvem/respondem às questões pendentes, em particular a questão que se encontre no topo da pilha (a questão “actual”), mas também as restantes questões da QUD (ex.: quando o utilizador responde antecipadamente a uma questão sem que essa questão lhe tenha sido colocada). Não havendo mais questões a esclarecer, o sistema terá obtido toda a informação que precisa para prestar o serviço. No estado final do diálogo a QUD encontra-se vazia e não existem mais questões.

As regras (com as respectivas condições e efeitos) dependem naturalmente da teoria de diálogo implementada e suportada pelo GD. As regras relacionadas com a teoria de diálogo QUD são especificadas de forma declarativa (e serão incluídas no conjunto de regras do DME), tal como é ilustrado no exemplo seguinte (uma regra com três condições), adaptado de [Kordoni *et al.*, 2007]:

Condição i) if user asks Q, push *respond*(Q) on AGENDA;

Condição ii) if *respond*(Q) on AGENDA and PLAN empty, find plan for Q and load to PLAN, e;

Condição iii) if *findout*(Q) first on PLAN, ask Q.

O significado destas três condições é o seguinte:

- i) Se foi identificada uma questão “Q” colocada pelo utilizador, o sistema deverá ter como prioridade responder a essa questão, pelo que a acção *respond*(Q) passa para a agenda do sistema;
- ii) Se há uma estratégia *respond* por executar e não existe um plano em execução, então é necessário tentar encontrar um plano que permita responder a “Q” e esse plano deve ser executado (pelo que é “carregado” na lista de planos), e;
- iii) Se a próxima estratégia do plano em execução for *findout*(Q) então o sistema deve despoletar um *move ask*(Q), apresentando a questão (ex. texto) ao utilizador.

O *Information State* é actualizado a cada passo.

3.6 Conclusão

As características do Midiki, nomeadamente por estar disponível em código aberto, fazem com que seja uma excelente ferramenta para experimentação e investigação nesta área. A liberdade de adaptação permite alterar ou adaptar o IS, os agentes ou qualquer das classes que compõem o Midiki.

Este factores tornam o Midiki um recurso muito conveniente para quem pretende aprender e experimentar (ie, praticar investigação) na área da gestão e dos gestores de diálogo, ou da abordagem ISU, ou da teoria de diálogo QUD (ou outras), ou em adaptações do *Information State* para interacção multimodal, qualquer que seja o nível de experiência e o interesse do investigador sobre estes temas.

Capítulo 4

Metodologia Proposta

4.1 Introdução

Este capítulo expõe uma metodologia para a modelação e prototipagem de diálogos no Midiki [Quintal & Sampaio, 2007]. A utilização da metodologia permite, com base no Midiki, a modelação de diálogos, construídos para ocorrer em determinado contexto, ou seja que pertencem a um determinado domínio, permitindo abordar tópicos e produzir resultados no âmbito desse domínio.

Embora o Midiki seja um sistema com um bom equilíbrio entre complexidade, flexibilidade e adaptabilidade a diferentes contextos (e totalmente disponível em Open Source), não existe documentação suficiente e adequada que explique de forma clara e intuitiva como criar e implementar diálogos no Midiki. De facto, a documentação disponibilizada não detalha suficientemente como criar diálogos com este gestor de diálogo. O manual do Midiki [Burke, 2005] discute aspectos de implementação do próprio gestor de diálogo e inclui várias referências técnicas de alguma complexidade, mas não indica ou exemplifica os passos a seguir. Os poucos exemplos apresentados no manual são sempre expressos em linguagem de programação Java. Isso faz com que seja difícil compreender como implementar novos diálogos com o Midiki, em particular por aqueles com menos experiência na temática dos sistemas de diálogo e

no conhecimento da linguagem Java. Em suma, o manual disponibilizado discute mais o próprio gestor de diálogo do que como utilizá-lo, não existindo outras referências de relevo nesse sentido. A metodologia agora proposta descreve os passos a seguir para a criação de diálogos com base no Midiki, sendo complementada por uma ferramenta de autoria para a sua prototipagem rápida, permitindo que mais pessoas possam criar diálogos com o Midiki, com menor esforço. Pelo facto de ser possível modelar diálogos utilizando apenas a ferramenta criada, são separados tanto quanto possível aspectos conceptuais de detalhes concretos das linguagens de implementação, facilitando muito a leitura, a portabilidade e a eventual reutilização de diálogos para diferentes contextos, por exemplo, fazendo uso da representação em XML proposta.

A fase inicial de um trabalho de modelação de um diálogo começa certamente numa análise mais clássica, de levantamento de requisitos, desenho e concepção de uma solução para a realização de uma tarefa no domínio em questão, tendo em conta os suportes tecnológicos a utilizar. Se recorrermos a um gestor de diálogo, isso quer dizer que uma das formas de levar a cabo a tarefa consistirá na realização de um diálogo entre um humano e uma máquina, o qual pode ocorrer apenas numa modalidade de interacção (diálogo unimodal) ou com suporte a várias modalidades (diálogo multimodal).

Pode acontecer que a maior parte do trabalho inicial, de levantamento de requisitos, até já tenha sido realizado previamente, como na situação em que apenas se pretenda acrescentar a possibilidade de interacção multimodal a um serviço já existente, ou seja a adição do suporte à interacção com base num sistema de diálogo poderá ocorrer sobre um sistema que já se encontre a funcionar com base noutro tipo de interfaces (ex.: interface web já existente, que se pretende estender com interacção em linguagem natural falada).

Não é essa fase inicial que constitui o âmbito deste trabalho. O âmbito deste trabalho enquadra-se numa fase que sucede essa fase inicial. Nessa fase posterior, na qual nos enquadrámos, estamos apenas preocupados em criar/implementar, sobre um gestor de diálogo, formas de interacção natural (em linguagem escrita ou falada) entre um humano e uma máquina com vista a levar a cabo determinada tarefa. Apresentamos neste capítulo a nossa metodologia para a criação de diálogos para o Midiki, através dos treze passos que a compõem.

Na secção 4.2 fazemos a apresentação das componentes de modelação de um diálogo; Na secção 4.3 é feita a apresentação e descrição dos passos da metodologia: secção 4.3.2: especificação do domínio: planos e atributos; secção 4.3.3: especificação do léxico (correspondência de *output matches* e *input matches*); secção 4.3.4: especificação das *cells* (Implementação de *handlers* e parametrização das

classes Tasks, DomainAgent e “domain executive”); Finalmente, na secção 4.3.5, são descritas as componentes procedimentais. O capítulo termina com uma conclusão, na secção 4.4.

4.2 Componentes da modelação do diálogo

No Midiki cada diálogo é modelado e realizado com base em planos compostos por um conjunto de *estratégias*. Por sua vez, cada *estratégia* pode decompor-se num número variável de acções elementares chamadas *dialogue moves* (ou simplesmente *moves*). Num diálogo falado, um *move* é equivalente a uma oração¹¹ (tomada isoladamente) de um participante, tendo em conta a sua caracterização semântica e lexical.

A metodologia considera dois tipos de componentes envolvidos na modelação e implementação de um diálogo: componentes *declarativas* e componentes *procedimentais*.

As componentes declarativas são três: o domínio (*domain*), o léxico (*lexicon*) e as *cells*¹². As componentes declarativas dizem respeito: i) à definição de planos e acções/estratégias num determinado contexto; ii) às expressões de entrada e de saída a utilizar (léxico) e iii) ainda às estruturas de dados (*cells*). Estas componentes são declarativas no sentido programático do termo, ou seja, a sua utilização no contexto da realização de um diálogo é uma responsabilidade do GD. A sua estrutura e composição é ilustrada na Figura 13. Por sua vez, o domínio é composto por três secções: 1) declaração de atributos; 2) definição de planos e 3) inicialização de questões. O léxico também é composto por três secções: 1) correspondência de *output matches*; 2) declaração de sinónimos e 3) correspondência de *input matches*. Uma *cell* compõe-se de quatro secções: 1) declaração de *slots*; 2) declaração de

¹¹ Por definição, em gramática, uma oração corresponde a cada um dos componentes maiores do período, formado por uma palavra ou conjunto de palavras com que se faz uma afirmação (ou proposição). Um período corresponde a uma frase organizada numa oração ou em orações [Infopedia, 2007b].

¹² Preferiu-se manter uma designação que fosse compatível com os termos utilizados na documentação do próprio Midiki, pelo que foi mantido o termo *cells* sem tradução. Uma *cell* define uma estrutura de dados que dá suporte à execução de planos no Midiki, providenciando um espaço de armazenamento e de ligação a serviços externos. É através das *cells* que são associados *slots* (espaços de armazenamento) e *handlers* às diferentes estratégias de um plano.

queries; 3) declaração de *methods*¹³ e 4) correspondência de *queries* e *methods* com os respectivos *handlers*.

Por outro lado, as componentes procedimentais dizem respeito a: i) à implementação dos *handlers* e a ii) à parametrização de código em classes próprias do Midiki, que devem ser invocadas na inicialização do diálogo. A execução de uma *query* ou de um *method* no Midiki corresponde à invocação do tipo *handler*, por exemplo para tratar de um acesso a uma base de dados ou outro tipo de serviços de *back end*. As componentes procedimentais são quatro e correspondem à implementação de código Java no sentido programático: i) implementação de *handlers*; ii) parametrização de tarefas (classe *tasks.java*); iii) parametrização do “*domain agent*” (classe *DomainAgent.java*) e iv) parametrização do “*dialogue manager executive*” (classe *dm.java*). As três classes referidas em ii) a iv) são geradas pela ferramenta.

4.2.1 Representação de alto nível em XML

A ferramenta desenvolvida para o suporte à autoria de diálogos permite a modelação dos diálogos através da especificação dos seus componentes declarativos.

Para as três componentes declarativas foram definidos os respectivos Document Type Definition (DTD), os quais são apresentados no Anexo I. Não foi implementada a funcionalidade que permita gerar ficheiros XML, embora essa funcionalidade pode ser implementada recorrendo à mesma tecnologia utilizada para gerar código Java (a Java Emitter Templates [Popma, 2004]), adaptando o código já existente.

Tanto o domínio como o léxico e as *cells* podem ser representados de forma independente da linguagem de programação final (Java, neste caso). Isso traz diversas vantagens, das quais se destaca:

- Facilidade de construção dos diálogos, nomeadamente o domínio, planos e léxico, recorrendo a uma ferramenta de modelação.
- Facilidade de leitura dos diálogos por outras pessoas e pelo próprio autor.
- Facilidade de conversão para representações em XML, compatíveis com outros gestores de diálogo, ou compatíveis com futuras versões do Midiki e independente da ferramenta de autoria utilizada.
- A independência face à linguagem de programação utilizada facilita e promove a reutilização.

¹³ O termo *method* tem um sentido semelhante ao que tem em programação Java e corresponde ao termo utilizado no manual do Midiki.

4.3 Apresentação e descrição dos passos da metodologia

Nesta secção são descritos em detalhe os treze passos que compõem a metodologia. Não existe uma sequência rígida de aplicação dos passos e o processo é necessariamente iterativo. A organização proposta para os diferentes passos apresenta a sequência que nos parece mais lógica e intuitiva. Certamente o passo que corresponde à construção de planos (o 1º passo) terá de preceder os restantes, mas um autor de um diálogo deverá realizar os passos descritos pela ordem que julgar mais conveniente. Os exemplos apresentados ao longo deste capítulo foram extraídos de um diálogo criado para um “agente de viagens” virtual, com o qual um cliente interage, em linguagem natural, para efectuar uma reserva de viagem.

4.3.1 Componentes declarativas (passos 1 a 9)

A especificação das componentes declarativas do diálogo constitui o trabalho de base de modelação do próprio diálogo, visto que é nesta parte que é representado aquilo que é pretendido que o sistema faça.

Recorrendo à ferramenta de autoria, o autor pode construir cada componente declarativa (domínio, léxico e *cells*) adicionando e parametrizando os elementos que a constituem. A ferramenta assiste o utilizador sempre que possível, em particular pela interligação de diferentes partes, evitando a duplicação de trabalho e reutilizando elementos já criados em passos anteriores.

O diagrama apresentado na Figura 13 mostra a estrutura dos elementos declarativos de um diálogo no Midiki e a relação de composição existente entre eles.

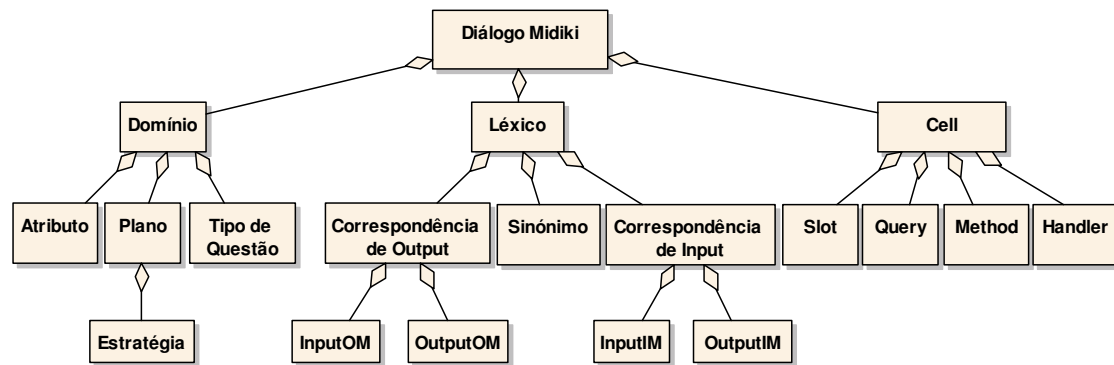


Figura 13: Os diferentes elementos que compõem a parte declarativa da implementação de um diálogo¹⁴

A equivalência da árvore anterior com os respectivos elementos XML é dada pela árvore apresentada na Figura 14.

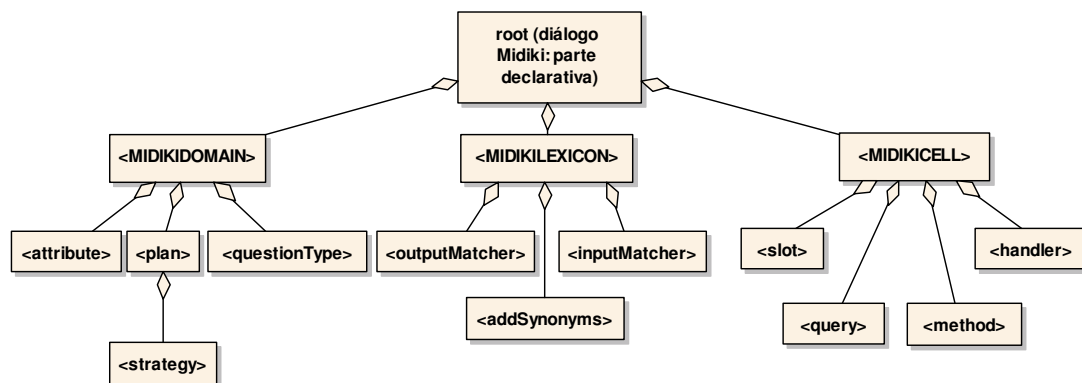


Figura 14: Tags XML para os elementos declarativos.

A Tabela 4 apresenta os nove passos que correspondem às três componentes declarativas da metodologia (domínio, léxico e *cells*).

Tabela 4: Lista de passos da metodologia e componentes declarativas em que se enquadram

Passo	Componente
Passo 1 - Construção de planos	Domínio
Passo 2 - Inicialização de atributos e de sinónimos	Domínio
Passo 3 - Inicialização de questões	Domínio

¹⁴ InputOM significa inputs para um OutputMatcher; OutputOM significa outputs de um OutputMatcher; InputIM significa inputs para um InputMatcher e OutputIM significa outputs de um InputMatcher.

Passo 4 - Correspondência de <i>output matches</i>	Léxico
Passo 5 - Correspondência de <i>input matches</i>	Léxico
Passo 6 - Declaração de <i>slots</i>	Cells
Passo 7 - Declaração de <i>queries</i>	Cells
Passo 8 - Declaração de <i>methods</i>	Cells
Passo 9 - Especificação de <i>handlers</i>	Cells

Estes nove passos são apresentados em detalhe nas secções seguintes.

4.3.2 Especificação do Domínio

A componente de domínio é descrita pelos três primeiros passos da metodologia. O DTD correspondente ao domínio é apresentado no Anexo I.

• Passo 1 - Construção de planos

O primeiro passo consiste em especificar as estruturas de planos. A construção de cada plano é feita pelo encadeamento de estratégias, constituindo o plano o elemento estrutural do diálogo. Um exemplo genérico (e simplificado) de um encadeamento de estratégias num plano é o seguinte: *i) findout(...) -> ii) queryCall(...) -> iii) inform(...)*, em que, neste caso: i) é colocada uma questão ao utilizador (uma estratégia *findout*); ii) é feito um processamento por forma a gerar um resultado relevante (no *handler* da *query* chamada pela *queryCall*) e iii) o utilizador é informado do resultado (com uma estratégia *inform*). Em XML corresponde ao elemento `<initializeTasks>`, do qual fazem parte os elementos `<plan>` e `<strategy>`. O bloco XML seguinte apresenta uma vista parcial de `<initializeTasks>`, com particular foco nos elementos `<strategy>` (as estratégias)¹⁵, que compõem o elemento `<plan>`. Um elemento `<baseCell>` indica em que *cell* se encontram os *slots* e os *handlers* para as estratégias deste plano. Apesar da especificação completa das *cells* se fazer apenas no passo 6, é necessário instanciar previamente uma *cell* (definindo o seu nome) para que possam ser criados novos planos. Este passo começa com a indicação da tarefa a executar por omissão (`<defaultTask>`). É a partir de `<defaultTask>` e de uma “*default question*”, que o sistema identifica por onde deve começar um diálogo. As estratégias incluídas neste bloco são *Findout*, *IfThen*, *MethodCall* e *Inform*:

```
<initializeTasks>
```

¹⁵ A utilização de XML nos exemplos permite apresentar cada parte de uma forma simples e fácil de ler, mantendo a sua semântica.

```

<defaultTask>reservarViagem</defaultTask>
  <plan>
    <name>reservarViagemPlan</name>
    <baseCell>reservarViagem</baseCell>
    <isSubPlan>no</isSubPlan>
    <strategy>
      <name>cidadeDestino</name>
      <type>Findout</type>
      <position>4</position>
    </strategy>
    <strategy>
      <name>pretendeRegresso</name>
      <type>Findout</type>
      <position>20</position>
    </strategy>
    <strategy>
      <name>IfThenStrategy</name>
      <type>IfThen</type>
      <guard>pretendeRegresso</guard>
      <value>sim</value>
      <subPlan>retornarSimPlan</subPlan>
      <position>24</position>
    </strategy>
    <strategy>
      <name>determinarCusto</name>
      <type>MethodCall</type>
      <baseCell>reservarViagem</baseCell>
      <position>60</position>
    </strategy>
    <strategy>
      <name>custoViagem</name>
      <type>Inform</type>
      <position>66</position>
    </strategy>
  </plan>
</initializeTasks>

```

O elemento `<plan>` apresentado diz respeito a um plano chamado “comprarViagemPlan”, do qual seleccionamos cinco estratégias. As estratégias do plano são iniciadas¹⁶ pela ordem dada pelo elemento `<position>`:

¹⁶ Note-se que uma *estratégia* não é executada sempre da mesma forma. Daí o utilizar-se o termo “iniciar” uma estratégia. A ordem pela qual a *estratégia* é executada não é necessariamente dada pela sua posição no plano (apesar da posição ser muito importante,

- Estratégias do tipo *findout* irão despoletar/desencadear um ou mais *moves* do tipo *ask*. Cada move do tipo *ask* faz com que o sistema apresente uma questão ao utilizador. Se a resposta do utilizador for errada ou mal interpretada pelo sistema, a pergunta inicial pode ser reformulada ao mesmo tempo que se dá ao utilizador alguma informação de ajuda. No Midiki são suportados dois níveis, em que: i) no primeiro nível é colocada a questão original e ii) caso haja necessidade de reformulação da questão original, a segunda mensagem é apresentada repetidamente até que o diálogo avance (ou seja abortado).
 - O texto para um *move* do tipo *ask* será obtido por consulta ao componente de léxico, procurando nos *output matches* por uma entrada da forma (“ask”, “nome-da-estratégia”);
 - A estratégia *IfThen* introduz uma chamada condicional a um subplano chamado “retornarSimPlan”, consoante a resposta do utilizador à questão desencadeada pela estratégia de *findout* anterior (pretendeRegresso, no exemplo da página 56). A condição subjacente ao *IfThen* é avaliada comparando o valor associado à estratégia indicada na guarda (em `<guard>pretendeRegresso</guard>`) com um valor indicado por `<value>`;
 - As estratégias *QueryCall* e *MethodCall* correspondem ao mecanismo disponibilizado pelo Midiki para o acesso a serviços externos (ex. acesso a uma base de dados). O elemento `<baseCell>` identifica a que *cell* está associada esta estratégia e consequentemente qual o seu *handler*;
 - A última estratégia apresentada no exemplo, do tipo *Inform*, produz uma informação de custo de uma viagem a ser apresentada ao utilizador. O seu valor deve ser definido pelo *handler* do *method* que lhe antecede. Note-se que é da responsabilidade do *handler* obter os parâmetros adequados, produzir o resultado pretendido e actualizar correctamente as *slots* que devem conter esse resultado.
-

uma estratégia pode até não ser executada se não for necessário). Além disso, não são conhecidos à partida exactamente quantos passos (*moves*) vão ocorrer até serem satisfeitas as necessidades de informação de uma estratégia. Uma questão pode ser colocada zero vezes, uma vez, duas vezes, ou mais, até que a resposta dada seja válida e aceite. Este facto resulta da forma como o próprio gestor de diálogo funciona, contribuindo para a produção de interacções mais naturais.

• Passo 2 - Inicialização de atributos e de sinónimos

Consiste em declarar, por enumeração, os diferentes tipos (ou categorias) de valores e, para cada tipo, quais os valores que pode tomar. Por exemplo, um tipo “*dias da semana*” poderá conter valores em {segunda-feira, terça-feira, quarta-feira, quinta-feira, sexta-feira, sábado, domingo}. Se uma estratégia *findout* (ou um qualquer *slot* de uma forma geral) estiver limitada a tomar valores do tipo “*dias da semana*” então passa a ser possível otimizar a forma de obter uma resposta do utilizador à questão associada a essa estratégia porque são conhecidos os termos possíveis para a resposta, nomeadamente através da sua validação e da disponibilização de *feedback* adequado quando essa resposta não é devidamente interpretada pelo sistema (no caso em que se esteja a utilizar o reconhecimento de fala, este é um aspecto crítico da interacção). Em termos de especificação XML corresponde ao elemento `<initializeAttributes>`. Por exemplo, um atributo com nome “metodosPagamento” com os valores {credito, multibanco, transferencia, cheque, dinheiro} será especificado em XML como se segue:

```
<initializeAttributes>
  <attribute>
    <attributeType>metodosPagamento</attributeType>
    <value>credito</value>
    <value>multibanco</value>
    <value>transferencia</value>
    <value>cheque</value>
    <value>dinheiro</value>
  </attribute>
</initializeAttributes>
```

Podemos definir sinónimos para estes valores ou para quaisquer atributos do domínio, conforme descrito a seguir.

Declaração de sinónimos

Podem ser definidas listas de sinónimos para qualquer um dos “valores” (ou termos) dos atributos. A criação de um conjunto de sinónimos para um termo corresponde ao elemento `<addSynonyms>`, onde se indica o tipo (com `<type>`), o termo base (com `<value>`) e os seus sinónimos (com um ou vários elementos `<synonym>`), agrupados num elemento `<word>`.

O elemento `<initializeSynonyms>` é incluído no código gerado para o léxico, embora a definição de sinónimos possa ser feita em dois momentos: no passo 2 e/ou

no passo 5. A definição de sinónimos simplifica a definição de atributos e permite baixar consideravelmente a quantidade de *input matches* necessários para interceptar diferentes formas como um utilizador se pode referir à mesma entidade ou à mesma acção. Por exemplo, para sinónimos do tipo "simNao", um sinónimo para "sim" pode conter os valores {correcto, exacto, obviamente} e um sinónimo para "não" pode conter os valores {nunca, impossível}, conforme apresentado a seguir, em XML. Os valores base do atributo "simNao" são apenas {sim, não}, enquanto os restantes termos constituem sinónimos destes dois termos de base (ou termos chave).

```
<initializeSynonyms>
  <addSynonyms>
    <type>simnao</type>
    <word>
      <value>sim</value>
      <synonym>certo</synonym>
      <synonym>correcto</synonym>
      <synonym>combinado</synonym>
      <synonym>exacto</synonym>
      <synonym>obviamente</synonym>
    </word>
    <word>
      <value>nao</value>
      <synonym>jamais</synonym>
      <synonym>nunca</synonym>
    </word>
  </addSynonyms>
</initializeSynonyms>
```

Poderíamos colocar todos estes termos como valores do tipo "simNao", mas a definição de sinónimos separada dos valores base estabelece, clarifica e simplifica a sua semântica. Neste caso, qualquer que seja o termo utilizado, a sua semântica reduz-se sempre a um "sim" ou um "não". Também podemos definir sinónimos para termos que possam ser indicadas em *inputs* do utilizador sem que a palavra base se encontre definida como um atributo.

Uma vez que a ferramenta recorre a uma base de dados para guardar toda a informação, será sempre possível recorrer a tabelas de informação adicionais, tais como um dicionário de sinónimos. Dessa forma, o autor poderia seleccionar de uma lista de sinónimos propostos pela ferramenta quais os que seriam incluídos na sua lista (ou a tabela de sinónimos poderia ser automaticamente consultada). Esta interessante funcionalidade poderá ser incluída em futuras versões da ferramenta.

• Passo 3 - Inicialização de questões

No último passo na especificação do domínio são associadas as estratégias do tipo *findout* às categorias de atributos definidos anteriormente (em `<initializeAttributes>`) como forma de delimitar os valores que poderão ser atribuídos como resposta do utilizador às questões desencadeadas por essas estratégias. A este passo corresponde o elemento XML `<initializeQuestions>` (inicialização de questões). Todas as estratégias que impliquem questões do sistema ao utilizador são incluídas na lista, juntamente com o respectivo tipo de resposta. Por exemplo, uma estratégia chamada "comoPagar" pode ser associada ao tipo "metodosPagamento" (pelo que apenas serão aceites como resposta a "comoPagar" valores pertencentes a esse tipo). O bloco XML seguinte exemplifica este passo.

```
<initializeQuestions>
  <questionType>
    <strategyName>cidadeDestino</strategyName>
    <attributeType>idades</attributeType>
  </questionType>
  <questionType>
    <strategyName>pretendeRegresso</strategyName>
    <attributeType>simnao</attributeType>
  </questionType>
  <questionType>
    <strategyName>comoPagar</strategyName>
    <attributeType>metodosPagamento</attributeType>
  </questionType>
</initializeQuestions>
```

Nomes dos ficheiros de domínio para as classes Java geradas:

Os elementos de domínio, criados nos passos 1, 2 e 3 são agregados numa classe Java do tipo *domain*. A classe do tipo *domain* terá o nome de domínio indicado pelo autor aquando da criação do projecto na ferramenta e terá a forma `<Nome_do_domínio>.java`.

4.3.3 Especificação do Léxico

A componente de léxico (ou *lexicon*) é descrita pelos passos 4 e 5. O DTD correspondente ao léxico é apresentada no Anexo I.

• Passo 4 – Correspondência de output matches

Consiste no mapeamento entre *dialogue moves* de saída do sistema, despoletados pela execução das estratégias, e texto a apresentar ao utilizador. É especificado na secção `<initializeOutputMatches>`, através de elementos `<outputMatcher>`. A correspondência tem a forma “Move de saída do sistema” → “Texto para o utilizador” (ex.: *ask(cidadeDestino)* → “Para que cidade pretende ir?”). Estas strings de texto são obtidas dos *output matches* incluídos no léxico.

O resultado da definição desta correspondência é particularmente relevante na execução de estratégias do tipo *findout* e do tipo *inform*. Como já foi referido, corresponde à especificação de quais as saídas que o sistema deve produzir quando há um *output* a fazer, dirigido ao utilizador. Pode ser feita a inclusão num *inform* de um valor gerado de forma a produzir uma saída composta com texto fixo e conteúdo variável. A parte variável da saída é normalmente obtida a partir do resultado de uma *query*.

No exemplo apresentado no segmento XML seguinte temos um *move* do tipo *ask*, relacionado com uma estratégia do tipo *findout* com o nome “cidadeDestino” e um *move* do tipo *inform* relacionado com uma estratégia do tipo *inform* com o nome “custoViagem”. Os elementos `<inConstruct>` e `<outConstruct>` correspondem aos construtores descritos no Anexo III.

```
<initializeOutputMatches>
  <outputMatcher>
    <inConstruct>InputNestedPredicate</inConstruct>
    <position>7</position>
    <inMoveType>ask</inMoveType>
    <inStrategy>cidadeDestino</inStrategy>
    <outText>Para que cidade pretende ir?</outText>
    <outConstruct>OutputTokens</outConstruct>
    <repeatMessage>Desculpe, nao reconheci a cidade para onde
pretende ir. Para saber a lista de cidades diga ajuda ou
opções.</repeatMessage>
  </outputMatcher>
  <outputMatcher>
    <inConstruct>InputNestedPredicate</inConstruct>
    <position>21</position>
    <inMoveType>inform</inMoveType>
    <inStrategy>custoViagem</inStrategy>
    <outText>A sua viagem custa </outText>
    <outConstruct>OutputVariable</outConstruct>
    <outText> Euros.</outText>
  </outputMatcher>
```

```
</initializeOutputMatches>
```

A descrição do bloco XML anterior é:

- O elemento `<position>` indica a posição (ou ordem) deste *output match* na lista de *outputMatchers*;
- O elemento `<outText>` indica a questão inicial a apresentar, enquanto o elemento `<repeatMessage>` indica a mensagem a apresentar caso a resposta à questão inicial não seja reconhecida como válida (o que se pode dever a várias razões). Esta mensagem não só deve reformular a questão, como deve também disponibilizar alguma ajuda ao utilizador. Neste exemplo concreto, se a resposta do utilizador à questão “Para que cidade pretende ir?” não for reconhecida como válida pelo sistema (por ex.: porque foi indicado um nome de uma cidade que não consta na lista de cidades ou o sistema reconheceu mal o nome proferido), será dado o seguinte feedback: “Desculpe, nao reconheci a cidade para onde pretende ir. Para saber a lista de cidades diga ajuda ou opções.” Assim que uma resposta do utilizador seja aceite como válida pelo sistema, o valor relevante do seu conteúdo, ou seja o nome de uma cidade, é associado à estratégia do tipo *findout* que deu origem à questão, sendo preenchido o respectivo *slot* com esse valor.
- O último *output match* deste exemplo faz corresponder a uma estratégia do tipo *inform* uma saída de texto para o utilizador onde é incluído um valor variável, com a forma “A sua viagem custa `<outConstruct> OutputVariable </outConstruct>` Euros.”, onde “OutputVariable” indica que deve ser inserido um valor nessa posição. A atribuição de um valor ao *slot* “custoViagem” é efectuada pelo *handler* de uma estratégia, do tipo *QueryCall* ou *MethodCall*, executada previamente no plano.

Tanto na correspondência de *output matches* (passo 4), como na correspondência de *input matches* (passo 5), é relevante a inclusão da posição de cada elemento `<outputMatcher>`, uma vez que o *parsing* de ambos é efectuado do “início para o fim” e é seleccionada a primeira ocorrência encontrada com correspondência válida.

Correspondência de output matches genéricos

Uma vez que existem algumas expressões que são suficientemente independentes do contexto e recorrentes em praticamente todos os diálogos, agrupá-mo-las num grupo genérico que é criado automaticamente para cada novo projecto. Devem depois ser

editados e actualizados de acordo com o diálogo implementado e com as preferências do seu autor. Estes elementos são descritos no Anexo III.

Por exemplo, um *move* de *output* do tipo *greet* pode corresponder, num diálogo A, a “Bem vindo à agência de viagens” e, num diálogo B, a “Bem vindo ao terminal Multibanco com interacção por fala”, mas corresponde sempre a uma mensagem de boas vindas. O mesmo se aplica à correspondência de *input matches* genéricos, relativos ao passo 5. Por exemplo, a expressão “bom dia” ou a expressão “olá”, dadas pelo utilizador ao sistema, pode corresponder sempre a um *move* de entrada do tipo *greet*.

• Passo 5 - Correspondência de input matches

Este passo está relacionado com o mapeamento entre entradas específicas de texto dadas pelo utilizador e *moves* específicos para o sistema. A correspondência é especificada através de elementos `<InputMatcher>`. As entradas podem ser expressões completas ou palavras-chave isoladas bem identificadas. Tomam a forma “Entrada de texto do utilizador” → “Move de entrada para o sistema”.

Por exemplo, um *input* de utilizador com a expressão “só ida” (considerada no contexto de um diálogo para a reserva de viagens, que temos vindo a utilizar) pode implicar, se inserida num *input match*, a geração de um *move* de entrada do tipo *answer*, o qual pode ser relacionado com uma estratégia de tipo *findout* chamada “pretendeRegresso” atribuindo-lhe um valor “não”. Supondo que existe pendente no plano uma questão “Pretende reservar também o regresso?” para a estratégia “pretendeRegresso”, a resposta à questão passa a ser “não” independentemente da questão já ter sido colocada ao utilizador ou não (foi dito que se pretende apenas uma viagem de ida, pelo que já não faz sentido perguntar se é pretendido o bilhete de regresso, tornando a interacção mais natural). Seria uma correspondência com a forma “só ida” → `answer("pretendeRegresso", "não")`. Como é de esperar, e no caso de ainda não ter ocorrido, já não ocorrerá a colocação de qualquer questão (que seria despoletada por essa estratégia), dado que já existe uma resposta válida avançada antecipadamente. Dessa forma, o sistema tem em conta e aproveita uma informação antecipada do utilizador.

Na representação destes mapeamentos utilizam-se os elementos XML `<InputMatcher>`. A utilização primária e principal dos `<inputMatcher>` será assim interceptar *inputs* típicos ou usuais do utilizador, esperados em situações e contextos concretos para os quais o diálogo tenha sido desenvolvido. Dessa forma, o sistema pode reagir de forma pré-definida (embora aparentemente natural) às expressões dos utilizadores, no sentido de otimizar a recolha de informação.

Associado à identificação de determinadas expressões pré-definidas, um *input match* permite também obter valores variáveis indicados pelo utilizador e associá-los a determinadas estratégias (em particular a estratégias do tipo *findout*), satisfazendo o sistema as suas necessidades informacionais sem que para tal seja necessário colocar questões específicas ao utilizador. Por exemplo, a entrada “Quero viajar no dia 12” indica, ao mesmo tempo, a intenção de “viajar” (o que permite identificar o tópico/tarefa a tratar) e que tal deve ocorrer no dia “12” (uma data). No exemplo em XML apresentado a seguir, ilustramos estes pormenores. Os tipos de argumentos dos elementos `<outConstruct>` são descritos no Anexo III.

No caso do segundo `<inputMatcher>` do bloco XML da página 64 (argumento `<position>` igual a 19), se o utilizador responder por exemplo “Quero partir no dia 12”, o valor 12 será associado à estratégia “diaPartida” (que é do tipo *findout*) da seguinte forma: Sendo detectada a (sub)expressão “partir no dia”, o sistema deverá procurar por um valor que possa atribuir a “diaPartida”. Se não for possível obter um valor que satisfaça as necessidades de informação da estratégia a pergunta deve ser novamente colocada ou refeita.

```
<initializeInputMatches>
  <inputMatcher>
    <inToken>só ida</inToken>
    <position>4</position>
    <outMove>answer</outMove>
    <outStrategy>pretendeRegresso</outStrategy>
    <outValue>não</outValue>
    <outConstruct>OutputNestedPredicate</outConstruct>
  </inputMatcher>
  <inputMatcher>
    <inToken>partir no dia</inToken>
    <position>19</position>
    <outMove>answer</outMove>
    <outStrategy>diaPartida</outStrategy>
    <outVar>yes</outVar>
    <outConstruct>OutputNestedPredicate</outConstruct>
  </inputMatcher>
</initializeInputMatches>
```

É importante referir que na decisão sobre as expressões ou palavras chave a incluir num `<inputMatcher>` (que compete ao autor do diálogo) tem de ser tido em conta que é necessário garantir que o sistema apenas deverá seleccionar uma expressão de forma unívoca, isto é, que será seleccionada a expressão que é pretendida e que para

aquela situação não é possível seleccionar outra expressão melhor. Sempre que uma expressão de um `<inputMatcher>` constitua uma subexpressão de outro `<inputMatcher>`, a mais pequena (ie, a subexpressão) deve ser inserida depois da maior, para que ambas possam ser utilizadas como e quando pretendido. Se a ordem for invertida, a expressão maior nunca será utilizada porque a sua subexpressão será sempre detectada primeiro (ex.: “pretendo só ida” deve ser inserido sempre antes de “só ida”, uma vez que ambas possuem a subexpressão “só ida”).

Nomes dos ficheiros de léxico, XML e Java:

Os elementos de léxico, criados nos passos 4 e 5 são agregados numa classe Java do tipo *lexicon*, juntamente com os sinónimos. A classe do tipo *lexicon* terá o nome indicado pelo autor para o léxico aquando da criação do projecto, na ferramenta. O nome tem a forma `<Nome_do_léxico>.java`.

4.3.4 Especificação das Cells

As estruturas de dados de suporte a um diálogo no Midiki são as *cells*. É a terceira e última componente declarativa do diálogo. O DTD correspondente às *cells* é apresentada no Anexo I. Na definição das *cells* de um projecto é necessário: i) especificar *slots* (e eventualmente o seu tipo) e ii) especificar *queries*, *methods* e os nomes das classes Java dos *handlers* para essas *queries* e *methods*. Uma estratégia do tipo *findout* terá necessariamente de possuir um *slot* onde serão guardado(s) o(s) valor(es) associados a essa estratégia, dado que um *findout* pressupõe a existência de uma resposta do utilizador a uma questão colocada pelo sistema, cujo valor deve ser guardado. No caso de uma estratégia *inform*, não será necessário associar um *slot* se a mensagem a apresentar cingir-se a uma *string* de texto fixa. Poderão existir ainda *slots* não associados a uma estratégia em particular, mas que servem para guardar o resultado de uma *query* ou *method*, o qual poderá ser depois utilizado de forma indirecta por uma estratégia do plano (por exemplo por um *IfThen*). Dizemos que se tratam de *slots* genéricos.

Um plano principal poderá possuir vários subplanos, mas todos terão como base a mesma *cell*. Um mesmo *handler* pode tratar uma ou várias *queries* ou *methods* (de forma análoga a uma função, que pode ser invocada de diversas formas e pode devolver diferentes resultados). Diferentes planos alternativos entre si, associados a tópicos distintos, podem e devem basear-se em *cells* distintas (e poderão naturalmente conter os seus próprios subplanos), com diferentes *slots* e com *handlers* distintos. Esta situação ocorre por exemplo quando inicialmente existem várias tarefas

possíveis. Uma vez identificada uma tarefa, esta é executada de forma independente das restantes tarefas, pelo que não existe necessidade de partilha de *cells*.

Note-se que, no código do Midiki, as *cells* constituem interfaces, concretizadas através de classes do tipo *Contract* (para o autor que analise o código Java correspondente à modelação de um diálogo é conveniente ter em conta esta relação).

• Passo 6 –Declaração de *slots*

Este passo consiste na declaração de uma lista de *slots* (e eventualmente do seu tipo), normalmente associados a estratégias, através da especificação do elemento `<addSlots>`. Quando são inicializadas as estruturas de dados, estas entradas permitem que sejam reservados os *slots* que irão receber a informação a associar às estratégias (em particular estratégias do tipo *findout*).

Os valores e tipos a incluir em `<addAttributes>` são aqueles que foram definidos em `<initializeAttributes>` (no passo 2 da metodologia), pelo que parte do passo 6 pode ser realizado de forma automática. Podem também ser declarados *slots* não associados a uma estratégia (*slot* genéricos). O conteúdo de um *slot* genérico, embora não associado a uma estratégia em particular, pode sempre ser limitado a determinado tipo. O bloco XML seguinte ilustra este passo:

```
<addSlots>
  <slot>
    <name>comoViajar</name>
    <type>meioDeTransporte</type>
  </slot>
  <slot>
    <name>pretendeRegresso</name>
    <type>simnao</type>
  </slot>
  <slot>
    <name>custoViagem</name>
    <type>notDefined</type>
  </slot>
</addSlots>
```

À estratégia chamada "comoViajar" associamos valores do tipo "meioDeTransporte" (o qual terá sido definido nos atributos do domínio), enquanto a "pretendeRegresso" apenas valores do tipo "simNao" (ex. "sim" ou "não"). No caso da estratégia "custoViagem", o seu tipo não é indicado pois não foi definido um atributo para esse tipo de valores (nesses casos, o tipo fica sempre indefinido).

O tipo de dados que se pode guardar num *slot* é bastante genérico. Internamente esses dados são guardados numa *collection* (Java), pelo que podem ser, por exemplo, números, strings ou mesmo listas.

• Passo 7 - Declaração de *queries*

Nesta secção são declaradas as estratégias do tipo *query* que fazem parte das acções dos planos. O elemento XML correspondente é `<addQueries>`. A atribuição de resultados da *query* a valores de estratégias (*slots*) fica à responsabilidade do respectivo *handler*. O resultado da *query* pode, por exemplo, ser apresentado em *moves* do tipo *inform*. O bloco XML seguinte ilustra a utilização de `<addQueries>`:

```
<addQueries>
  <query>
    <strategy>obterLugaresDisponiveisAviao</strategy>
    <descr>Indicar ao cliente quais os lugares disponíveis no
avião.</descr>
  </query>
</addQueries>
```

• Passo 8 - Declaração de *methods*

De forma análoga às *queries* no passo anterior, nesta secção são indicados os *methods* que fazem parte das acções dos planos associados à *cell* em questão. A associação de resultados entre um *method* e uma estratégia é feita de forma idêntica às *queries*. O elemento XML correspondente é `<addMethods>`.

A especificação de uma *query* ou de um *method* é muito semelhante. Podemos considerar que se utilizará uma *query* para aceder a bases de dados e *methods* para tratar outro tipo de serviços. O exemplo seguinte ilustra a especificação de `<addMethods>`.

```
<addMethods>
  <method>
    <strategy>determinarCustoViagem</strategy>
    <descr>Calcular custo da viagem de acordo com as preferências
do cliente.</descr>
  </method>
</addMethods>
```

• Passo 9 – Especificação de handlers

Este passo consiste na criação de uma lista de *handlers*, através da indicação dos nomes das classes Java que implementam o código para *queries* ou *methods*. Um *handler* é uma classe Java que implementa uma função ou um serviço e é chamado sempre que é executada uma *query* ou um *method* num plano. Corresponde à especificação do elemento XML `<initializeHandlers>`. Como exemplo, uma *query* chamada "obterLugaresDisponiveisAviao" pode ter como *handler* uma classe Java com o nome "lugaresDisponiveisAviaoHandler.class", pelo que esta classe será invocada sempre que num plano surja a estratégia `QueryCall(obterLugaresDisponiveisAviao)`.

```
<initializeHandlers>
  <handler>
    <strategyName>obterLugaresDisponiveisAviao</strategyName>
    <handlerType>query</handlerType>
    <javaClass>lugaresDisponiveisAviaoHandler</javaClass>
  </handler>
  <handler>
    <strategyName>determinarCustoViagem</strategyName>
    <handlerType>method</handlerType>
    <javaClass>custoViagemHandler</javaClass>
  </handler>
</initializeHandlers>
```

Nomes dos ficheiros de cells, XML e Java:

Os elementos de uma *cell*, criados nos passos 6 a 9 são agregados numa classe Java do tipo *cells* e são gerados pela ferramenta. O ficheiro do tipo *cells* terá o nome indicado pelo autor aquando da criação da *cell*, com a forma `<Nome_da_cell>.java`.

4.3.5 Componentes procedimentais (passos 10 a 13)

Após a conclusão dos passos 1 a 9 (que pode ser iterativa), segue-se a fase de implementação dos componentes procedimentais (naturalmente também iterativa). Inclui-se aqui o passo 10 (implementação de *handlers*); o passo 11 (configuração da classe *Tasks.java*); o passo 12 (configuração da classe *domainAgent.java*) e o passo 13 (configuração da classe "dm.java", onde a classe "dm" se designa por "domain manager executive"). Apenas no caso da implementação dos *handlers* (passo 10) será necessário trabalho de programação, caso se pretenda efectuar, por exemplo, acesso a bases de dados ou implementar algoritmos específicos para tratamento dos dados,

enquanto que nos passos 11 a 13 as classes são automaticamente parametrizadas e geradas. Esses quatro passos são descritos de seguida.

- **Passo 10 – Implementação de handlers**

As classes Java que implementam as funções ou serviços de tratamento de acesso à informação de *back end* (tipicamente acesso a bases de dados) são os chamados *handlers*. Sempre que uma estratégia do plano do tipo *queryCall* ou *methodCall* seja encontrado, é invocado o respectivo *handler*. Numa situação típica, o utilizador será informado do resultado da *query* ou do *method* através de um *move* subsequente do tipo *inform*, cujo *slot* guarda o resultado da *query* ou *method*. No *handler*, uma vez definido o resultado, deve ser feita uma procura de um *slot* com o identificador pretendido para que esse valor seja associado à estratégia com esse nome (todos os *slots* acessíveis num *handler* são incluídos no *template* desse *handler*, que é gerado pela ferramenta). No *template* são dadas indicações sobre como obter parâmetros (ie, como obter valores guardados em *slots*) e como devolver resultados (ie, como definir o valor de *slots* que deverão conter o resultado).

Para além dos *templates* gerados, o código de um *handler* é necessariamente específico e deve ser escrito caso a caso, consoante a lógica de negócio a implementar. Parâmetros de acesso a bases de dados tais como o nome da BD; a sua localização (URL); *passwords*; conectores; etc, devem ser tidos em conta e necessariamente incluídos no código pelo programador. Os nomes a atribuir às classes do tipo *handler* devem ser os que foram definidos no passo 9 (especificação de *handlers*).

- **Passo 11 - Parametrização da classe Tasks**

Na classe *Tasks.java* é incluída informação de suporte à execução do diálogo através da instanciação das diferentes *cells* que fazem parte das tarefas do diálogo (uma classe *cell* é instanciada através de uma classe *contract* Midiki). O código para “Tasks.java” é gerado pela ferramenta e o nome da classe é fixo.

- **Passo 12 - Parametrização da classe DomainAgent**

Na classe *DomainAgent* é incluída informação de inicialização de *cells* (incluindo *cells* internas ao próprio Midiki) e de *handlers* e é estabelecida a ligação de vários componentes do diálogo ao *information state*, nomeadamente a ligação do léxico e do domínio ao IS.

O código Java relativo a esta classe é gerado pela ferramenta e o nome da classe é sempre “DomainAgent.java”.

• **Passo 13 - Parametrização da classe “domain executive”**

Este passo consiste na parametrização da classe “*dialogue manager executive*” (ou simplesmente “dm”). A classe “*dm.java*” (neste caso o nome classe não tem de ser fixo, pelo que lhe chamamos genericamente “dm”), é responsável pela inicialização dos vários agentes e interfaces do sistema (ex.: a janela de interacção por texto). É incluída nesta classe uma referência aos agentes que compõem o gestor de diálogo para cada projecto criado. A inclusão de algumas funcionalidades de *debugging* disponibilizadas pelo Midiki é também feita aqui (por omissão essas funcionalidades estão activas sempre que o código é compilado, podendo ser facilmente desactivadas comentando as respectivas linhas, com // ou /*...*/). Essas funcionalidades de *debugging* são: i) possibilidade de inclusão do *DomainSpy*, o qual permite visualizar em tempo de execução o estado do *information state* e ii) visualização da estrutura de planos do GD. O código Java relativo a esta classe é gerado pela ferramenta.

Após a compilação de todas as classes geradas e criadas, com o conjunto do Midiki, é esta classe que deve ser invocada para executar um diálogo. O nome do ficheiro Java da classe “dm” pode ser qualquer.

4.4 Conclusão

Neste capítulo foi apresentada uma metodologia para a modelação e prototipagem de diálogos no Midiki. Quer a metodologia, quer a ferramenta que a suporta constituem contributos novos para a autoria de diálogos com o gestor de diálogo Midiki. Nesse sentido realçamos os seguintes pontos:

- A metodologia traduz e apresenta de forma simples e intuitiva um assunto normalmente de abordagem complexa e pouco acessível à partida;
- A metodologia é suportada por uma ferramenta de autoria que não só torna a concepção e construção dos diálogos mais simples, como também automatiza a maior parte do trabalho necessário na criação de todas as classes (Java) que permitem compilar o código fonte obtido para que possa ser executado;
- A partir dos DTDs propostos poderá ser gerado código XML para a representação das componentes declarativas do diálogo com todas as vantagens daí decorrentes (ex.: facilidade de leitura e conversão). A geração de XML não foi implementada na versão actual da ferramenta, mas poderá ser

utilizada o mesmo código base em JET (*Java Emitter Templates*), já utilizado para a geração de Java.

Capítulo 5

Implementação da Ferramenta de Autoria

5.1 Introdução

A ferramenta implementada visa dar suporte à criação e prototipagem rápida de diálogos para o Midiki com base na metodologia apresentada, abstraindo o mais possível a modelação do diálogo da sua codificação em Java e automatizando o maior número possível de passos. Neste capítulo descrevemos os aspectos mais relevantes da implementação da ferramenta. Na secção 5.2 é feita uma descrição geral da ferramenta; Na secção 5.3 é feita a apresentação dos casos de uso; A secção 5.4 corresponde à descrição da arquitectura e componentes mais importantes; Na secção 5.5 são apresentadas as principais funcionalidades, interfaces de utilizador e ecrãs; Na secção 5.6 expomos a justificação para as opções tecnológicas tomadas; A secção 5.7 indica os elementos do diálogo suportados; Finalmente, a secção 5.8 apresenta as ferramentas utilizadas no desenvolvimento da aplicação, terminando o capítulo na secção 5.9 com os principais passos a seguir na instalação da ferramenta de autoria.

5.2 Descrição geral

A ferramenta é disponibilizada como uma aplicação *web*. Aspectos como a possibilidade de acesso remoto para diferentes utilizadores e o funcionamento transparente sobre diferentes plataformas e sistemas operativos justificam essa opção. A interface de utilizador funciona sobre um *browser* standard. Recorrendo à ferramenta, um autor pode criar e editar diferentes projectos. Um projecto corresponde à modelação de um diálogo para determinado domínio.

Toda a informação manipulada pela aplicação é guardada numa base de dados. Um utilizador acede à ferramenta efectuando *login* e pode criar e gerir vários projectos.

Em consonância com a metodologia, um projecto deve compor-se de três partes ou componentes: o domínio (*domain*), o léxico (*lexicon*) e as *cells*. Para cada componente deve ser possível adicionar e parametrizar os elementos que o constituem. A ferramenta suporta o utilizador sempre que possível, automatizando procedimentos e eliminando duplicação de trabalho.

O autor pode compilar o código gerado (com as restantes fontes do Midiki) sem qualquer modificação (com a excepção dos *handlers*, que devem ser sempre concluídos “manualmente”). O autor pode obviamente optimizar e estender o código obtido antes de efectuar a compilação.

A interface está implementada em inglês por uma questão de divulgação e a designação que damos em inglês à ferramenta é *Midiki Authoring Tool* (MAT). Na próxima secção são apresentados e descritos os casos de uso relativos à utilização da ferramenta por parte de um autor de um diálogo.

5.3 Casos de uso

A Figura 15 apresenta os principais casos de uso relacionados com a ferramenta, com relevância para “criar projecto/diálogo” e para a geração de classes.

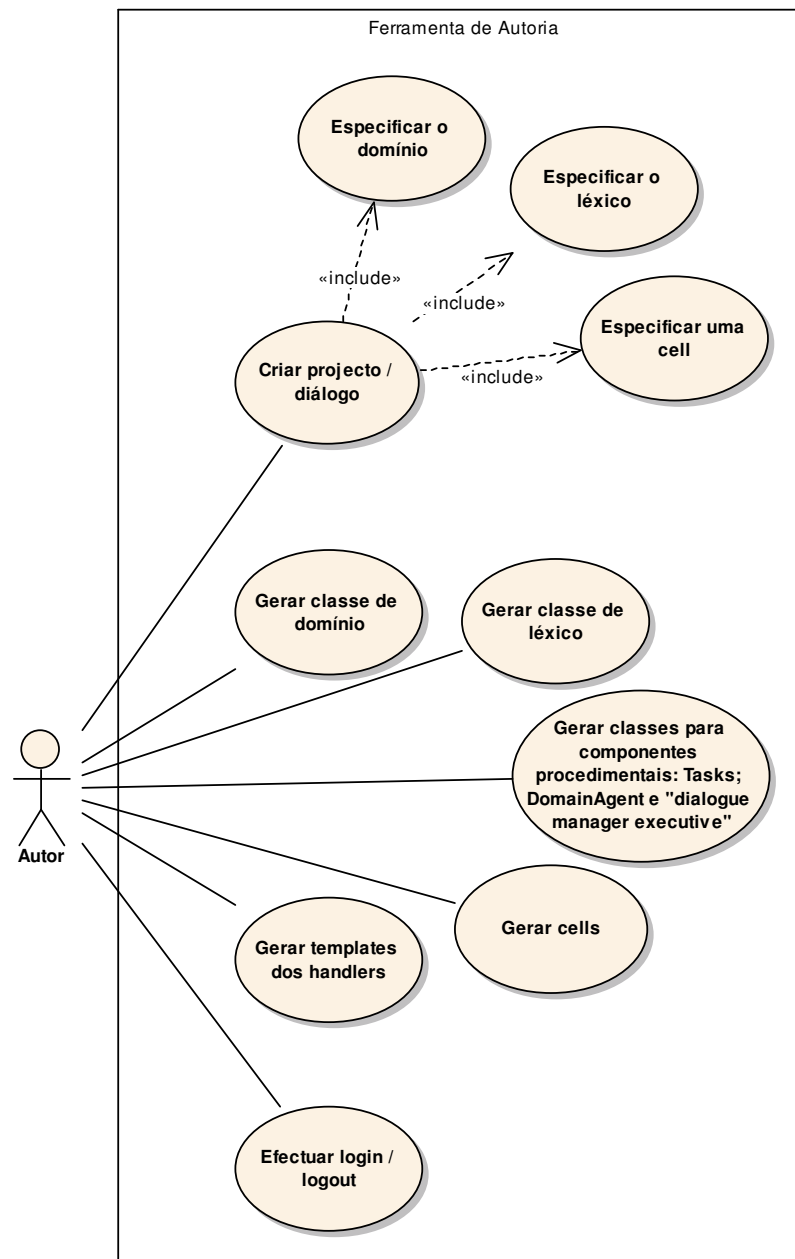


Figura 15: Principais casos de uso para a ferramenta.

O caso “criar projecto/diálogo” é detalhado na Figura 16, com destaque para a especificação de cada componente declarativa.

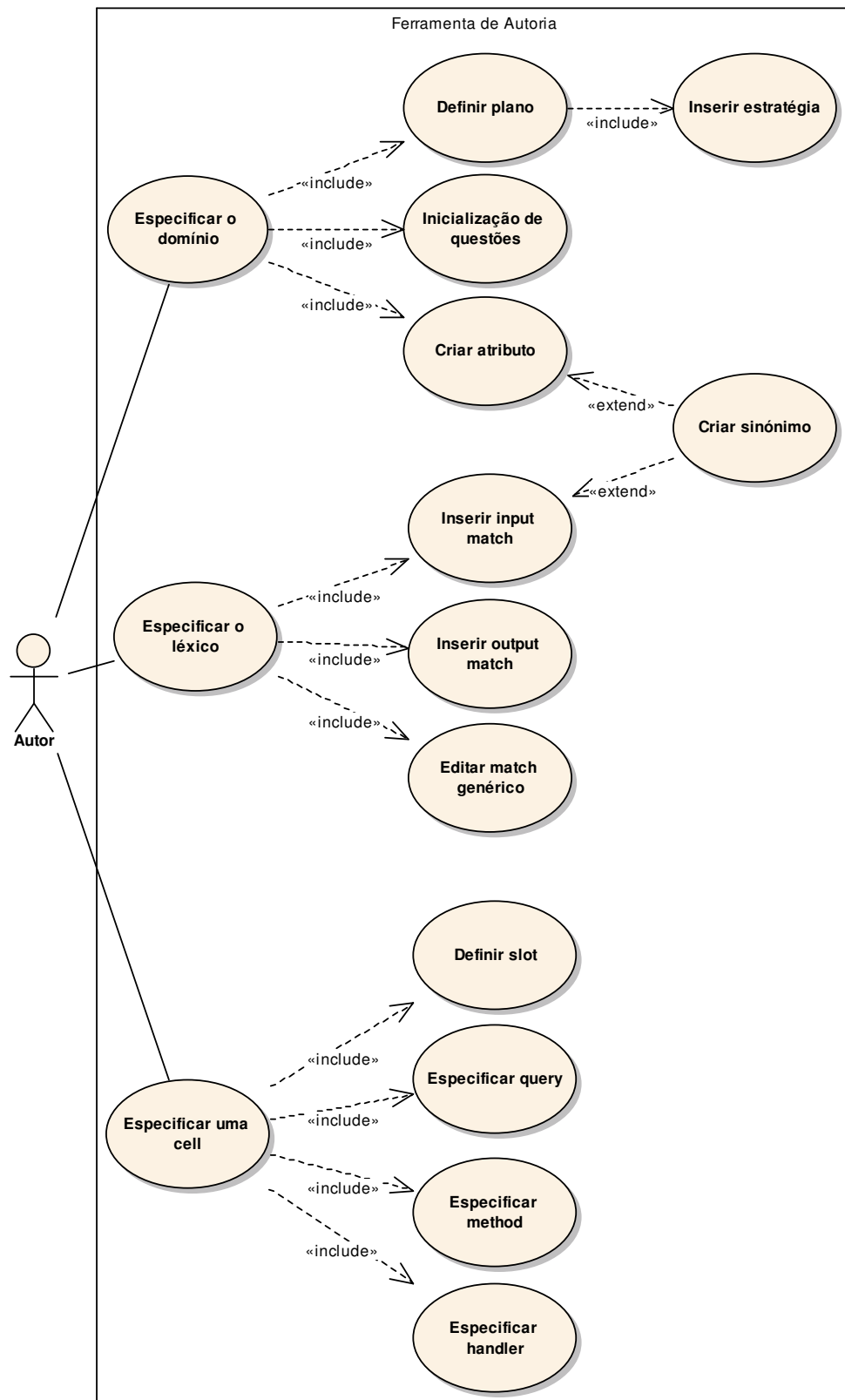


Figura 16: Decomposição/detalhe do caso de uso “criar projecto/diálogo”

Efectuar *login/logout*: Permite aceder à ferramenta a partir de um browser web/http standard, indicando um *nome-de-utilizador* e uma *password* disponibilizados.

Criar projecto/diálogo: Permite instanciar um novo diálogo. A modelação de um diálogo começa com a criação de um projecto e compõe-se da especificação dos três componentes declarativos (domínio, léxico e *cells*), cujos casos de uso são descritos à frente. A completa modelação de um diálogo corresponde a um projecto completo na ferramenta.

Especificar o domínio: Consiste em construir a componente de domínio através da criação de planos, inclusão de estratégias nos planos, definição de atributos e inicialização de questões.

Definir plano: Permite criar um novo plano para o diálogo/projecto actual.

Inserir estratégia: Permite inserir uma nova estratégia no plano actual (plano seleccionado). Começa pela selecção do tipo de estratégia a incluir e consoante o tipo de estratégia pretendido são solicitados os campos necessários à sua caracterização. O autor pode confirmar a inserção ou cancelá-la.

Criar atributo: Permite criar tipos de valores. Cada tipo possui um identificador e uma lista de valores que constituem esse tipo.

Inicialização de questões: Permite indicar qual o tipo de atributo para cada estratégia do tipo *Findout* (cada *Findout* vai corresponder a uma questão do sistema para o utilizador do diálogo¹⁷).

Especificar o léxico: Consiste em construir a componente de léxico através da criação de *input matches* e *output matches*.

Inserir *output match*: Permite definir o mapeamento entre *dialogue moves* de saída do sistema e texto a apresentar ao utilizador. O autor começa pela selecção do tipo de *output match*. Consoante o tipo de *output match* escolhido, são solicitados os campos necessários à sua caracterização. O autor pode confirmar a inserção ou cancelá-la.

¹⁷ Nalgumas partes deste texto utilizamos a referência “utilizador do diálogo” para nos referirmos a um utilizador final do sistema de diálogo a implementar, ou seja o utente, ou cliente, a quem se destina o sistema de diálogo que venha a ser criado.

Inserir *input match*: Permite definir o mapeamento entre entradas específicas de texto, dadas pelo utilizador do diálogo, e *moves* específicos para o sistema. O autor começa por indicar qual a expressão de entrada e depois indica que tipo de *move* deve o sistema executar sempre que essa expressão (texto) seja detectada num *input* do utilizador do diálogo. O autor pode confirmar a inserção ou cancelá-la.

Criar sinónimo: Permite criar sinónimos para os dos tipos definidos em “criar atributo”, mais precisamente para os valores que compõem esses tipos.

Editar *match* genérico: Permite que o autor indique o texto a apresentar para alguns *output* e *input matches* que são comuns a qualquer diálogo. Estes *matches* possuem sempre a mesma semântica, mas o seu léxico varia consoante o contexto do diálogo. Esses *matches* genéricos são automaticamente criados para cada novo projecto e o que o autor tem de fazer é verificar e/ou editar o texto que deve ser mostrado. Estes elementos são descritos no Anexo III.

Especificar uma *cell*: Permite instanciar uma *cell*. Qualquer plano, *query* ou *method* serão sempre baseados numa *cell*. Fazem parte da especificação completa da *cell* a especificação dos respectivos *slots*, *queries*, *methods* e *handlers*. Inicialmente, uma *cell* pode ser instanciada definindo apenas o seu nome e posteriormente pode ser completada com os elementos que a compõem.

Definir *slot*: Permite definir uma lista de *slots*, os quais irão receber a informação a associar às estratégias (em particular estratégias do tipo *Findout* e *Inform*).

Especificar *query*: Permite adicionar a uma *cell* a declaração de uma *query*, a qual estará associada a uma estratégia pertencente a um dos planos baseados na *cell* em questão.

Especificar *method*: Permite adicionar a uma *cell* a declaração de um *method*, o qual estará associado a uma estratégia pertencente a um dos planos baseados na *cell* em questão.

Especificar *handler*: Permite que o autor indique os nomes das classes Java que implementam o código para as *queries* ou *methods* dos planos do diálogo.

Gerar classes para componentes procedimentais: Permite obter as classes Java de *Tasks*, *DomainAgent* e *dmExecutive*. Gerar classe *Tasks.java* permite obter a classe *Tasks.java*; Gerar classe *DomainAgent.java* permite obter a classe *DomainAgent.java* e Gerar classe *dmExecutive.java* permite obter a classe “domain executive” (o nome desta classe pode variar).

Gerar classe de domínio: Permite obter a classe Java que implementa o domínio do diálogo.

Gerar classe de léxico: Permite obter a classe Java que implementa o léxico do diálogo.

Gerar *cells*: Consiste em gerar a(s) classe(s) Java que implementa(m) as *cells* de suporte à implementação do diálogo. O autor deve seleccionar cada *cell* que tenha criado e gerar o seu código, uma de cada vez.

Gerar *templates dos handlers*: Consiste em gerar a(s) classe(s) Java que disponibiliza(m) o(s) *handler(s)* declarados. O autor deve seleccionar cada *handler* que tenha declarado e gerar o código do respectivo *template*, um de cada vez.

5.4 Arquitectura

A concepção/desenho da aplicação *web* implementada teve como referência o padrão MVC, com um nível de apresentação, um nível de lógica de negócio e um nível de dados. O seu funcionamento recorre ao modelo cliente-servidor e a estruturação de todas fontes seguiu uma estrutura modular. O padrão MVC [Wikipedia, 2007] é uma recomendação para arquitecturas de software que defende a separação de uma aplicação em três camadas: Model (Modelo); View (Vista) e Controller (Controlador). Isto é feito de tal forma que quaisquer alterações numa das partes (componentes) possam ser feitas em qualquer altura sem que isso implique alterações/impacto nas outras partes.

5.4.1 Componentes da arquitectura

O nível de apresentação é suportado por um browser (cliente fino/leve) que acede ao servidor por HTTP (sobre uma ligação TCP/IP). O nível de lógica de negócio é suportado pelo servidor aplicacional ZOPE-2 [Fulton, 1998]; [Fulton, 2005] com as diversas classes implementadas em Python [van Rossum, 1991] e em Java. O nível de

dados é suportado por uma base de dados relacional (mysql). A Figura 17 ilustra o padrão MVC implementado.

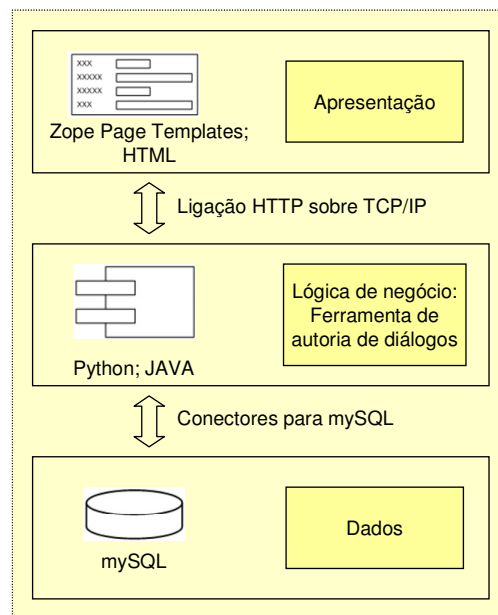


Figura 17: Padrão MVC implementado

O diagrama da Figura 18 detalha os principais componentes da ferramenta e o modo de interacção entre eles, numa perspectiva mais detalhada do padrão MVC implementado.

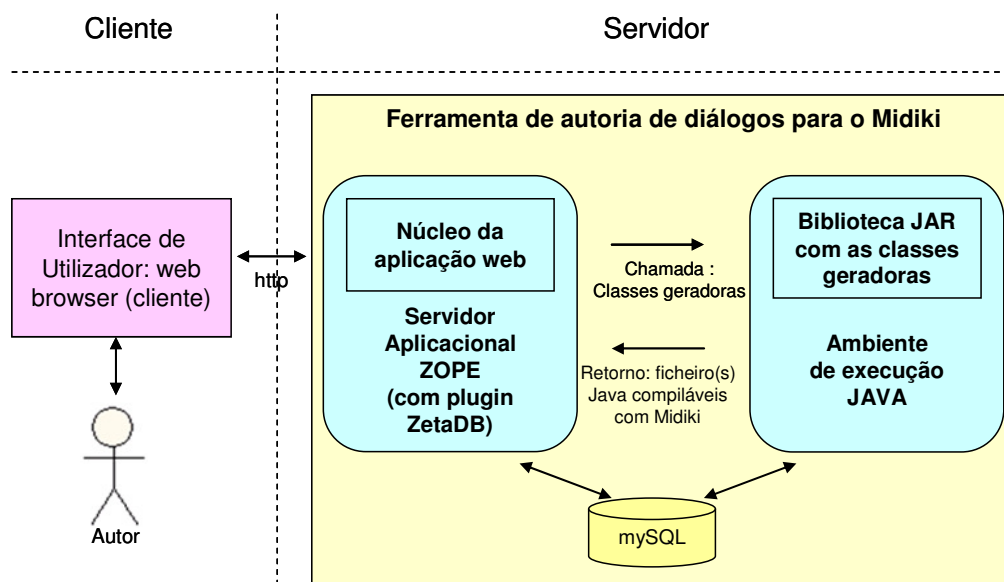


Figura 18: Componentes da ferramenta de engenharia/autoria de diálogos

O ambiente de execução da ferramenta é o ZOPE, o qual inclui um servidor web integrado e apresenta mecanismos próprios para a geração de páginas HTML dinâmicas (as ZOPE Page Templates ou ZPT). Um documento ZPT pode ser editado com um editor HTML standard, permitindo um tratamento separado de cada parte do desenvolvimento (apresentação, lógica de negócio e dados). O ZOPE integra um interpretador Python e possui conectores para os principais sistemas de gestão de bases de dados, nomeadamente MySQL. Permite igualmente a ligação a funcionalidades externas através da invocação indirecta de classes Python externas, as quais permitem aceder, por exemplo a bibliotecas/pacotes JAR. Sobre o ambiente ZOPE de base instalamos o *product* ZetaDB [Camps, 2003]. Os *products* no ZOPE correspondem a módulos (ou *plugins*) que adicionam funcionalidades extra ao Zope. O ZetaDB facilita o desenvolvimento de aplicações web, sobre Zope, baseadas em dados. A ligação da lógica de negócio ao nível de dados é efectuada sobre conectores ZOPE e Java (para MySQL).

A estrutura de pastas e objectos de “código fonte” criados reflecte igualmente a separação modular e por níveis MVC: Cada funcionalidade principal disponível na aplicação final possui uma correspondência com uma determinada parte do código fonte. Por sua vez, cada uma dessas partes (ou módulos) apresenta e distingue claramente os objectos que dizem respeito à apresentação, à lógica de negócio ou aos dados.

5.4.2 A base de dados que suporta a ferramenta

A base de dados armazena toda a informação sobre os diversos projectos em implementação. Inclui ainda tabelas de suporte e gestão (ex.: utilizadores e *passwords*, etc). As tabelas que compõem a base de dados relacional são sumariamente descritas nos parágrafos seguintes.

A tabela “**project**” guarda informação de gestão dos vários projectos/diálogos que sejam criados. Um projecto será sempre detido por um utilizador. Um projecto serve de referência para agrupar toda a informação de um diálogo tal que, um diálogo está relacionado com um projecto de forma biunívoca. Para cada projecto existe apenas um domínio e um léxico e podem existir várias *cells* e *handlers*.

A tabela “**cell**” guarda informação relativa às *cells* criadas. Uma *cell* pertence a um projecto e um projecto poderá incluir várias *cells*.

A tabela “**cellattribute**” (corresponde aos *slots*) guarda informação relativa aos nomes dos *slots* a criar e tipos de atributos que podem conter.

A tabela “**cellquery**” guarda informação relativa à associação entre estratégias do tipo *QueryCall* e as *queries/handlers* que deverão implementá-las.

A tabela “**cellmethod**” guarda informação relativa à associação entre estratégias do tipo *MethodCall* e os *methods/handlers* que deverão implementá-las.

A tabela “**handler**” contém a indicação dos nomes dos *handlers* de cada *query* ou *method*. O nome de um handler corresponde a uma classe Java chamada para tratar uma *query* ou um *method*.

A tabela “**domain**” guarda informação relativa ao domínio de um diálogo. É ao nível do domínio que são criados os planos.

A tabela “**plan**” guarda informação relativa aos planos criados. Um plano pertence a um domínio. O detalhe do plano em termos de estratégias será dado por registos na tabela “*strategy*”.

A tabela “**attribute**” guarda informação de base de um atributo do domínio. A lista de valores de cada atributo é guardada na tabela “**attributevalue**”.

Na tabela “**strategy**” cada registo guarda informação relativa a uma estratégia de um determinado plano.

A tabela “**strategytype**” guarda informação de referência sobre os tipos de estratégias suportados pela ferramenta. Os tipos de estratégias são descritos no Anexo II.

A tabela “**questiontype**” guarda informação relativa à associação entre estratégias do tipo *Findout* e tipos de atributos para um determinado domínio/diálogo.

A tabela “**movetype**” guarda informação de referência sobre os tipos de *dialogue moves* suportados pela ferramenta. Os tipos de *dialogue moves* são descritos no Anexo II.

A tabela “**lexicon**” guarda informação relativa ao léxico de um projecto. É ao nível do léxico que são criados *input* e *output matches*.

Na tabela “**outputmatch**” cada registo descreve um *output match* num diálogo.

Na tabela “**inputmatch**” cada registo descreve um *input match* num diálogo.

A tabela “**inomconstructtype**” guarda informação de referência sobre os tipos de construtores de entrada para os *output matches* suportados pela ferramenta. Esses tipos de construtores são descritos no Anexo III, embora apenas sejam relevantes ao nível do código Java pois são manipuladas de forma transparente pela ferramenta.

A tabela “**outomconstructtype**” guarda informação de referência sobre os tipos de construtores de saída para os *output matches* suportados pela ferramenta. Esses tipos de construtores são descritos no Anexo III.

A tabela “**outimconstructtype**” guarda informação de referência sobre os tipos de construtores de saída para os *input matches* suportados pela ferramenta. Esses tipos de construtores são descritos no Anexo III.

A tabela “**synonym**” guarda informação da definição de sinónimos. Cada sinónimo é guardado num registo da tabela “**synvalue**”.

Na tabela “**generics**”, para cada projecto, são guardadas strings de texto a serem utilizadas nos *input* ou *output matches* genéricos.

O modelo entidade-relação da base de dados é apresentado no no Anexo V.

5.5 Interfaces de utilizador e ecrãs

Nesta secção são apresentadas e descritas as principais interfaces (ecrãs) e funcionalidades disponibilizadas pela ferramenta. Passamos a apresentar a organização das interfaces:

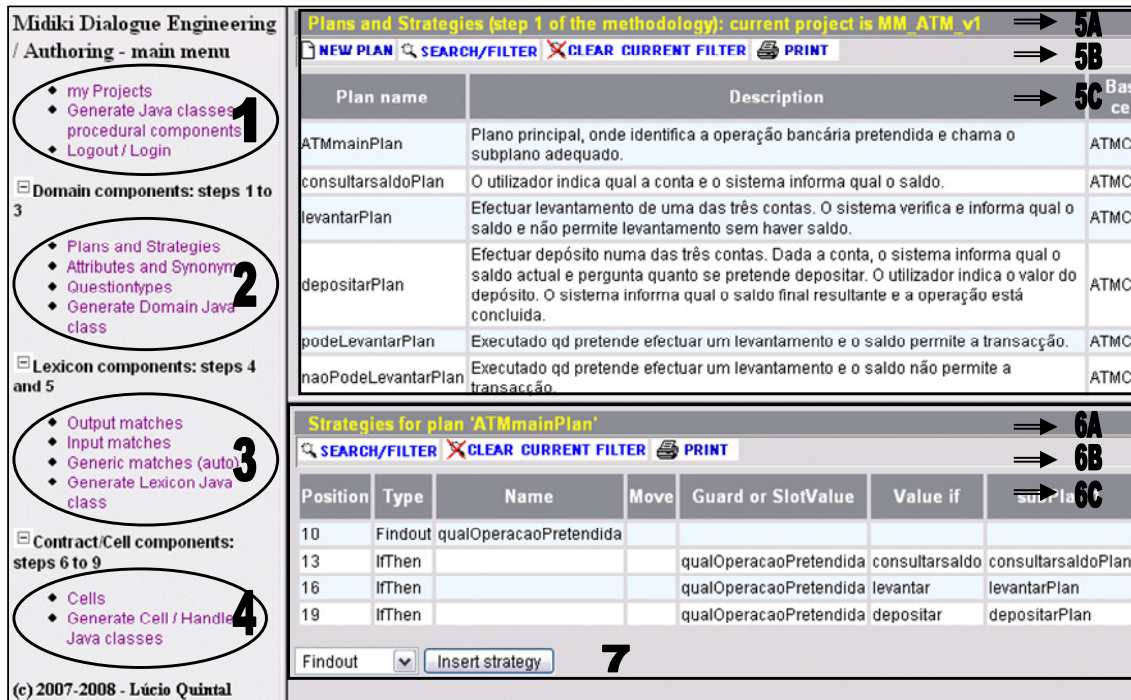


Figura 19: Organização geral da interface da ferramenta

Um ecrã divide-se em duas ou três partes distintas: o menu principal encontra-se sempre do lado esquerdo (regiões 1 a 4 na Figura 19). O lado direito corresponde à área de visualização e edição. Por omissão os elementos são apresentados em lista (ex. lista de planos de um projecto). Seleccionando uma das linhas, o item em questão é aberto em modo de edição. Nalguns casos, a *frame* do lado direito pode subdividir-se em duas partes (uma parte superior e uma parte inferior) quando existam elementos compostos por listas de outros elementos (por exemplo os planos, que são compostos por um conjunto de estratégias e onde, para cada plano são listadas na parte inferior as respectivas estratégias). As regiões 1 a 4, assinaladas na Figura 19, dizem respeito às seguintes funcionalidades: região 1) listagem de projectos; geração de classes Java para as componentes procedimentais e login/logout; região 2) Funcionalidades relacionadas com a componente domínio (passos 1 a 3 da metodologia); região 3) Funcionalidades relacionadas com a componente léxico (passos 4 e 5 da metodologia); região 4) Funcionalidades relacionadas com a componente *cells* (passos 6 a 9 da metodologia).





As regiões 5A e 6A identificam cada vista, indicando qual o elemento seleccionado e qual o passo da metodologia abordado. As regiões 5B e 6B correspondem à barra de botões / sub-menus disponíveis. As regiões 5C e 6C apresentam os cabeçalhos de cada lista de elementos. O menu de selecção e inserção de estratégias, de *input matches* e *output matches* surge no fim de cada lista, na parte assinalada na Figura 19 com o nº 7.

Uma sessão termina ao fazer-se *logout* ou ao fim de 20 minutos de inactividade. A ferramenta suporta apenas um utilizador de cada vez.

Algumas funcionalidades genéricas das interfaces:

As funcionalidades descritas nesta secção aplicam-se a qualquer ecrã. No modo de listagem, qualquer lista pode ser ordenada, quer por ordem ascendente, quer por ordem descendente. Clicando no cabeçalho da respectiva coluna/campo é feita uma ordenação ascendente (o cabeçalho apresenta-se em letra branca sobre fundo cinzento – regiões 5C e 6C). Clicando novamente no cabeçalho a ordenação é feita por ordem inversa, de forma alternada.

Sempre que uma lista ultrapassa um determinado número máximo de linhas, as restantes linhas são mostradas noutra página. Os botões para navegar entre páginas surgem quando apropriado e são os seguintes:

-  **START** Saltar para as primeiras entradas
-  **PREVIOUS** Saltar para a página anterior
-  **NEXT** Saltar para a página seguinte
-  **END** Saltar para as últimas entradas

Se o utilizador pretender eliminar um elemento da lista, abre-o para edição e selecciona “DELETE” (é sempre feita uma confirmação de eliminação). Para imprimir uma lista ou um formulário pode ser utilizada a opção “PRINT” (corresponde a imprimir a janela actual do browser). Quando aplicável, existe um botão “BACK”, que permite voltar ao ecrã anterior sem que sejam gravadas eventuais alterações. Se for clicado o botão “Save”, as alterações são gravadas e a aplicação volta ao modo de listagem inicial, mostrando eventuais elementos novos.

Nalguns ecrãs é possível pesquisar/filtrar elementos através da opção “SEARCH/FILTER” (por exemplo para listar apenas determinados tipos de estratégias). Após a aplicação de um filtro de pesquisa é necessário seleccionar “CLEAR CURRENT FILTER” para voltar a visualizar a lista completa, ie, para limpar o filtro aplicado.

Nas secções seguintes descrevemos as funcionalidades disponíveis para cada ecrã.

Começar um projecto para a criação de um diálogo

A criação de um novo diálogo implica executar os passos descritos a seguir, sendo que os dois primeiros dizem sobretudo respeito à utilização da ferramenta de autoria. Os restantes correspondem aos passos da metodologia:

- Criar um projecto
- Instanciar o nome de uma *cell*
- Criar planos e estratégias (1º passo da metodologia)
- Definir atributos e sinónimos (2º passo da metodologia)
- Inicializar questões do(s) plano(s) (3º passo da metodologia)
- Fazer correspondência de *output matchers* (4º passo da metodologia)
- Fazer correspondência de *input matchers* (5º passo da metodologia)
- Definir *slots* (6º passo da metodologia)
- Declarar *queries* do plano (7º passo da metodologia)
- Declarar *methods* do plano (8º passo da metodologia)
- Especificar *handlers* (9º passo da metodologia)
- Gerar código para as *cells* e *templates* para os *handlers*
- Gerar código para os componentes procedimentais

Para começar a criar um diálogo, o autor tem de começar por criar um projecto, o qual servirá de referência a todas as partes do diálogo. Para criar um projecto deve seleccionar “my Projects -> NEW PROJECT”.

Project name	Author	Remarks	Domain	Lexicon
viagens1	Lúcio	First example created / generated with the tool.	click here	click here
AreasVolumes2	Lúcio	This is the case study being implemented: version two implements more Contracts, separating main plans	click here	click here
areaRectangulo	Lúcio	tests	click here	click here
MM_ATM_v1	Lucio	Simular um acesso a uma caixa multibanco (ATM), suportando as seguintes quatro operações: i) Consultar saldo de uma conta; ii) Efectuar levantamento; iii) Efectuar depósito e iv) Transferir dinheiro.	click here	click here
AgenteViagens	Lúcio Quintal	Diálogo de demonstração, que implementa um agente de viagens virtual, com interacção em linguagem natural, falada ou escrita.	click here	click here

AgenteViagens

Figura 20: Lista de projectos de um autor

Cada utilizador tem acesso à sua lista de projectos. Para definir um projecto como o projecto actual, o utilizador pode usar a opção “Set Current Project” ou clicar num dos links para o domínio ou para o léxico na lista de projectos (em “click here”). Por omissão as classes do tipo *domain* e *lexicon* recebem um nome composto pelo nome do projecto e um sufixo *Domain* ou *Lexicon*, respectivamente. O autor pode alterar

esses nomes se o pretender. A Figura 21 ilustra a edição dos campos que identificam um projecto.

Projects	
BACK PRINT DELETE	
Project name:	AgenteViagens
Author:	Lúcio Quintal
Remarks:	Diálogo de demonstração, que implementa um agente de viagens virtual, com interacção em linguagem natural, falada ou escrita.
Domain:	click here
Lexicon:	click here
<input type="button" value="Save"/>	

Figura 21: Criação/Edição de campos relativos a um projecto

Para iniciar a criação do diálogo, é igualmente necessário criar pelo menos uma *cell* que possa servir de base aos planos a criar a partir do passo 1 da metodologia. Inicialmente deverá apenas ser indicado o nome da *cell*. Os restantes campos da *cell* (*slots*, *queries*, etc) devem ser criados apenas mais tarde, quando todos os planos e estratégias do diálogo já estiverem criados. A Figura 22 ilustra a edição dos campos que definem uma *cell*.

Cells	
BACK PRINT DELETE	
Cell name:	AgenteViagens
Description:	Esta é a cell de suporte aos planos do projecto 'AgenteViagens'
Cell slots:	click here
Cell queries:	click here
Cell methods:	click here
Cell handlers:	click here
<input type="button" value="Save"/>	

Figura 22: Criação/Edição de elementos relativos a uma *cell*

Criação de planos e estratégias (1º passo da metodologia):

Seleccionando “Plans and Strategies”, o utilizador acede à lista de planos. Clicando num dos campos de uma linha relativa a um plano, o utilizador pode editar os campos que caracterizam esse plano. Num diálogo existirá normalmente um plano principal e

existirão subplanos associados ao plano principal. Para criar um novo plano o autor usa a opção "NEW PLAN" do menu. Para cada plano deverá indicar qual a sua *cell* de suporte.

Plans and Strategies (step 1 of the methodology): current project is AgenteViagens	
NEW PLAN SEARCH/FILTER CLEAR CURRENT FILTER PRINT	
Plan name	Description
PlanoPrincipal	Identificar o tópico a tratar e chamar um subplano adequado: 'obter informações' ou 'efectuar reserva'.
OpcoesRegressoPlan	

Figura 23: Criação de novo plano - opção "NEW PLAN"

A Figura 24 ilustra os campos que caracterizam um plano.

Plans	
BACK PRINT DELETE	
Plan name:	PlanoPrincipal
Description:	Identificar o tópico a tratar e chamar um subplano adequado: 'obter informações' ou 'efectuar reserva'.
Base cell:	AgenteViagensContractID
Is subPlan?:	<input type="checkbox"/>
Strategies:	click here
<input type="button" value="Save"/>	

Figura 24: Edição de elementos que caracterizam um plano

Depois de criado um plano, o utilizador pode adicionar-lhe estratégias. A listagem ou criação de estratégias começa sempre com a selecção de um plano da lista de planos do diálogo. As estratégias pertencentes ao plano seleccionado são listadas na parte inferior do ecrã (numa *frame* distinta). A opção "Insert strategy" é incluída no final da lista de estratégias e serve para inserir uma nova estratégia no plano actual, conforme ilustrado pela Figura 25.

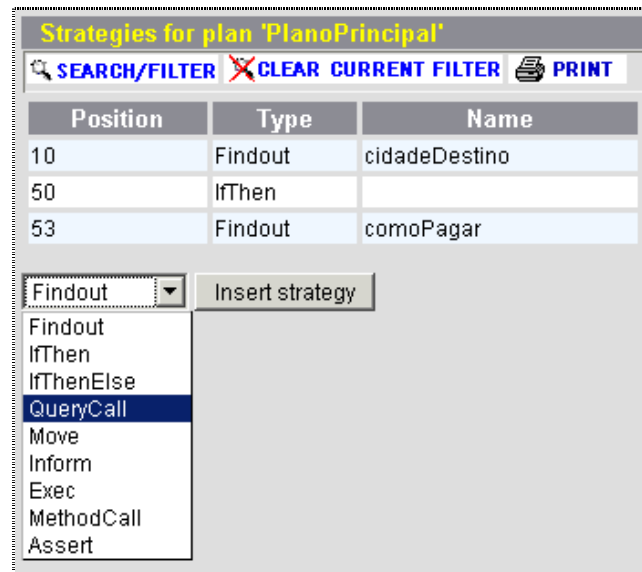


Figura 25: Vista parcial de uma lista de estratégias de um plano, com destaque para a opção de inserção de novas estratégias

Uma vez seleccionado o tipo de estratégia a inserir e carregando em “Insert strategy” surge o ecrã de criação da estratégia. A Figura 26 ilustra a inserção de uma estratégia do tipo *Findout*:

Strategies: Editing strategy in plan 'PlanoPrincipal'	
BACK PRINT DELETE	
Position:	10
Type:	Findout
Name:	cidadeDestino
Description:	Obter cidade de destino.
Save	

Figura 26: Criação/Edição de uma estratégia do tipo *Findout*

A Figura 27 ilustra a inserção de uma estratégia do tipo *IfThen*:

Strategies: Editing strategy in plan 'PlanoPrincipal'	
<div> BACK PRINT DELETE </div>	
Position:	50
Type:	IfThen
Description:	Se o valor de 'pretendeRegresso' for 'sim' é necessário saber data e hora pretendidos para o regresso.
Guard or Slot value:	pretendeRegresso
Value if:	sim
subPlan-1:	OpcoesRegressoPlan
<div>Save</div>	

Figura 27: Criação/Edição uma estratégia do tipo *IfThen*

A Figura 28 apresenta o ecrã de filtragem/pesquisa de estratégias. No exemplo mostrado, o filtro “Type” foi definido como “Findout”, pelo que seriam listadas apenas estratégias desse tipo no plano actual, após carregar-se em “Send”.

Strategies	
<div> BACK CLEAR CURRENT FILTER </div>	
Position:	
Type:	Findout
Name:	
Move:	Any
Description:	
Guard:	
Value if:	
subPlan-1:	Any
subPlan-2:	Any
<div>Send</div>	

Figura 28: Ecrã de pesquisa/filtragem de estratégias

Note-se que nos ecrãs de definição de filtros estão disponíveis todos os campos que caracterizam as diferentes estratégias, sendo que nem todos esses campos dizem respeito a todos os tipos de estratégias (por ex. o campo “Guard” só se aplica a estratégias dos tipos *IfThen* e *IfThenElse*).

Criação de atributos e sinónimos (2ª passo da metodologia):

Seleccionando “Attributes and Synonyms” o utilizador pode consultar, criar ou apagar atributos. Para cada atributo o utilizador tem a possibilidade de criar uma lista de sinónimos (tal como descrito na metodologia). A Figura 29 ilustra a criação de atributos, dos seus valores e de eventuais sinónimos.

The screenshot shows a web interface titled 'Attributes'. It has a header with 'BACK', 'PRINT', and 'DELETE' buttons. Below the header, there are input fields for 'Attribute type' (containing 'metodosPagamento') and 'Description' (containing 'Opções de pagamento a disponibilizar'). There is a link 'click here' for 'Attribute values' and a 'Save' button. Below this, there is a section titled 'Attributes and synonyms (step 2 of the methodology): current project is AgenteViagens'. This section has a 'NEW ATTRIBUTE VALUE' button and a 'PRINT' button. It contains a table with three columns: 'Value', 'Synonyms', and 'Attribute type'.

Value	Synonyms	Attribute type
multibanco	click here	metodosPagamento
dinheiro	click here	metodosPagamento
cheque	click here	metodosPagamento
credito	click here	metodosPagamento
transferencia	click here	metodosPagamento

Figura 29: Criação/Edição de atributos

Na parte inferior surge a lista de valores do atributo e os link para a criação/edição de eventuais sinónimos de cada um desses valores

No exemplo da Figura 29, o atributo “metodosPagamento” está definido com os valores {multibanco, dinheiro, cheque, credito, transferência}. Para cada um desses valores podemos definir uma lista de sinónimos (seleccionando “click here” na coluna “synonyms”). Podem ser definidos sinónimos para atributos do domínio ou para expressões que ocorram em *input matches*. Relativamente à definição de atributos e sinónimos, ter em conta os seguintes aspectos:

- Todos os atributos devem ser preferencialmente criados em minúsculas, sem acentos e sem espaços entre partes de um nome composto (ex.: “Nova lorque” não deve constituir um atributo de base, ao passo que “Nova-lorque” é aceite. A expressão “Nova lorque” apenas pode ser reconhecida se for definida como um sinónimo de, por exe., “Nova-lorque”;
- De forma a facilitar a verificação de correspondência entre termos indicados pelo utilizador e termos detidos pelo sistema, todos os *inputs* de um utilizador

durante o diálogo são sempre convertidos em minúsculas antes de serem processados internamente;

- Relativamente aos acentos, podem ser criadas palavras com acentos nas listas de sinónimos, de forma que o utilizador possa indicá-las (ex.: incluir “opções” como um sinónimo de “opcoes” ou “não” como sinónimo de “nao”). Termos sem acentos devem ser incluídos como atributos e termos com acentos devem ser incluídos como sinónimos;
- Atributos cujos valores base sejam numéricos devem ser definidos como tal, podendo-se então criar listas de sinónimos para esses atributos, nos quais podem ser descritos por extenso. Por exemplo, um atributo com os valores {1, 2, 3, 4, 5} pode ter como sinónimos {um, dois, três, quatro, cinco}, mas o conjunto de valores base que compõem esse tipo é o conjunto {1, 2, 3, 4, 5} e não o conjunto {um, dois, três, quatro, cinco};
- Ao ser criado um novo projecto, é inserido um atributo genérico denominado “generalAttributeForSynonyms”, que pode ser utilizado para criar termos base para palavras ou expressões que não estejam associadas a atributos. Por exemplo as expressões {bom dia, boa tarde, boa noite, olá} podem ser sinónimas de “ola”, onde “olá” é um atributo do tipo “generalAttributeForSynonyms” (termo base).

Inicialização de questões do(s) plano(s) (3º passo da metodologia):

O último passo na especificação do domínio consiste em criar uma lista de questões dos planos. Para cada estratégia do tipo *Findout*, deve ser indicado o seu tipo. Ao clicar em “NEW QUESTION TYPE” o autor deve seleccionar uma estratégia (da lista apresentada) e indicar o seu tipo (de entre os tipos definidos nos atributos). A Figura 30 apresenta o ecrã para a opção “NEW QUESTION TYPE”.

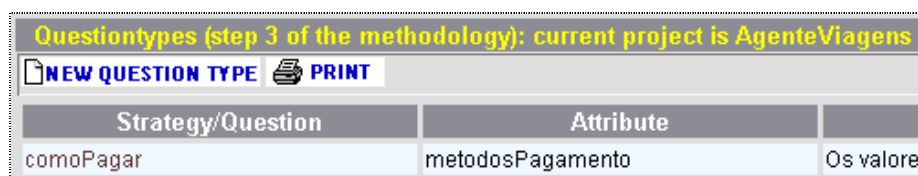


Figura 30: Inicialização de questões (criação de *question types*)

A Figura 31 ilustra os campos que caracterizam uma “Question type”.

Figura 31: Edição de *question types*

Nas secções anteriores descrevemos as principais funcionalidades e ecrãs associados à especificação do domínio (passos 1 a 3 da metodologia). Nas secções seguintes são descritas as funcionalidades e ecrãs relativos à especificação do léxico (passos 4 e 5).

Correspondência de *Output Matchers* (4º passo da metodologia):

Cada *output match* ou *input match* possui uma entrada (aquilo que recebe) e uma saída. Um *output match* recebe como entrada um *dialogue move* gerado pelo GD e tem como saída uma expressão textual dirigida ao utilizador do diálogo. No seu conjunto, um *output match* corresponde a um *output* do sistema para o utilizador. Por sua vez, um *input match* recebe como entrada uma expressão textual dada pelo utilizador do diálogo (ex. uma oração proferida) e gera como saída um *dialogue move* dirigido ao GD. No seu conjunto, um *input match* corresponde a um *input* do utilizador para o sistema.

Ao seleccionar a opção “Output matches” no menu principal, o autor pode definir o mapeamento entre *moves* específicos de saída do sistema e strings de *output* para o utilizador do diálogo. A Figura 32 ilustra como é definido/especificado um *output match*.

Position	Strategy	Inform value	Main output text	
10	cidadeDestino		Para que cidade pretende ir?	Desculpe, não rec

In: (Findout strategy name) -> Out: (Text string; Text string if answer error)
 In: (Findout strategy name) -> Out: (Text string; Text string if answer error)
 In: (Inform strategy name) -> Out: (Text string)
 In: (Inform strategy name) -> Out: (Text string1; Variable name,[Optional text string2])
 In: (Inform strategy name; value) -> Out: (Text string)

Figura 32: Inserção de *output matches*

O formato de apresentação de cada um dos quatro tipos de *output match* que podem ser definidos segue a seguinte estrutura: A parte de *input* (representada por “In:”) indica para que tipo de estratégia pretendemos definir o *output match* (um *Findout* ou um *Inform*) e quais os argumentos a indicar. A parte de *output* (representada por “Out:”) indica a forma como pretendemos compor a saída de texto para o utilizador do diálogo. Na Tabela 5 são descritos os tipos de *output match* que podem ser definidos:

Tabela 5: Tipos de *output matches* que podem ser definidos com a ferramenta

Tipo de output match	Descrição
In: (Findout strategy name) Out: (Text string; Text string if answer error)	Esta opção destina-se a definir texto de saída para estratégias <i>Findout</i> . In: o autor deverá seleccionar na lista apresentada uma estratégia <i>Findout</i> . Out: indicar o texto a apresentar ao utilizador do diálogo como questão para esse <i>Findout</i> e o texto a apresentar caso a resposta do utilizador não seja reconhecida e a questão deva ser repetida.
In: (Inform strategy name) Out: (Text string)	Esta opção destina-se a definir texto de saída para estratégias <i>Inform</i> “simples”. In: o autor deverá seleccionar na lista apresentada uma estratégia <i>Inform</i> . Out: indicar o texto (fixo) a apresentar ao utilizador do diálogo.
In: (Inform strategy name) Out: (Text string1; Variable name,[Optional text string2])	Esta opção destina-se a definir texto de saída para estratégias <i>Inform</i> “compostas”, juntando ao texto fixo uma parte variável. In: o autor deverá seleccionar na lista apresentada uma estratégia <i>Inform</i> . Out: indicar a parte fixa do texto a apresentar ao utilizador do diálogo, sendo que neste caso haverá uma parte variável que será incluída entre as partes fixas 1 e 2. A parte 2 (string2) é opcional. A parte variável corresponde a um valor que tenha sido atribuído ao <i>slot</i> desta estratégia, o que em geral ocorre no <i>handler</i> de uma <i>query</i> ou <i>method</i> .
In: (Inform strategy name; value) Out: (Text string)	Esta opção destina-se a definir texto de saída para estratégias <i>Inform</i> , consoante o valor da estratégia de entrada. In: o autor deverá seleccionar na lista apresentada uma estratégia <i>Inform</i> e indicar o valor que esta deverá conter para que seja gerada a saída (a estratégia será executada apenas se o respectivo <i>slot</i> contiver o valor indicado). Out: indicar o texto fixo a apresentar ao utilizador do diálogo.

O autor selecciona da lista o tipo de *output match* a criar e escolhe “Define output match” (são suportados quatro tipos de *output match*: um para estratégias *Findout* e três para estratégias *Inform*). A Figura 33 ilustra a especificação de um *output match* do tipo *Findout* para uma estratégia com nome “cidadeDestino”. O texto incluído no campo “Main output text” corresponde à pergunta principal e o texto incluído no campo “Text to output if user answer error” corresponde à mensagem a apresentar no caso da resposta à pergunta principal não ser reconhecida como válida.

Output matches	
<div> <div>BACK</div> <div>PRINT</div> <div>DELETE</div> </div>	
Output match position:	10
Strategy:	cidadeDestino
Main output text:	Para que cidade pretende ir?
Text to output if user answer error:	Desculpe, não reconheci a cidade para onde pretende ir. Para saber a lista de cidades diga ajuda ou opções.
<div>Save</div>	

Figura 33: Edição de *ouput matches*

Note-se que não são suportadas pela ferramenta todas as variantes de *input* e *output matches* possíveis no Midiki, em particular no tipo de argumentos que podem ser incluídos (construídos necessariamente através de código Java). São sim suportadas as formas de *input* e *output matches* mais utilizadas e mais importantes.

Correspondência de *Input Matchers* (5º passo da metodologia):

Ao seleccionar a opção “Input matches” no menu principal, o autor pode associar expressões específicas de entrada, isto é, *inputs* do utilizador do diálogo. O principal interesse dos *inputMatchers* é interpretar palavras ou expressões típicas dos utilizadores, expectáveis em situações concretas e próprias do domínio em questão, e assim permitir que o sistema reaja de forma predefinida a essas expressões típicas. A Figura 34 ilustra como é definido ou especificado um *input match*.

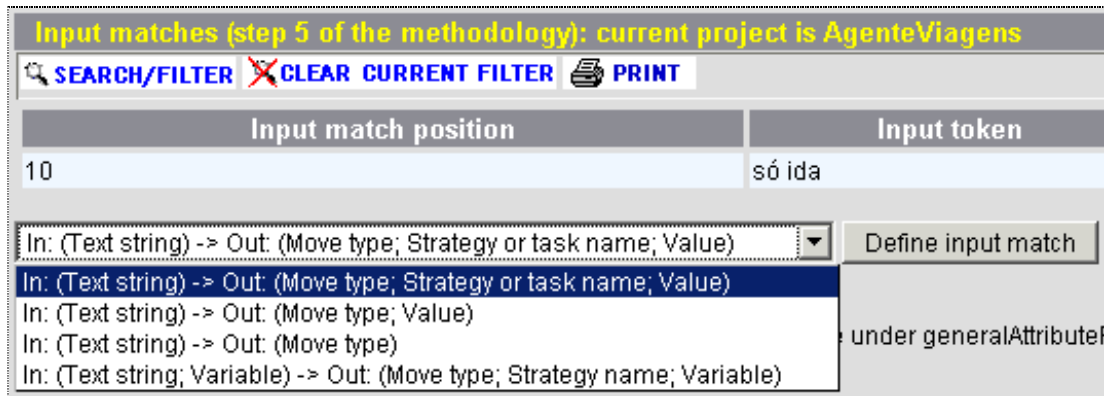


Figura 34: Inserção de *input matches*

O formato de apresentação de cada um dos quatro tipos de *input match* que podem ser definidos segue a seguinte estrutura: A parte de *input* (representado por “In: Text string”) corresponde a uma expressão de texto a ser detectada nos *inputs* do utilizador do diálogo. A parte de *output* (representado por “Out:”) indica a forma como pretendemos compor o *dialogue move* resultante para o sistema de diálogo. Na Tabela 6 são descritos os tipos de *input match* que podem ser definidos (todas as opções referidas nesta tabela permitem gerar um *dialogue move* para o sistema a partir de um *input* do utilizador do diálogo):

Tabela 6: Tipos de *input match* que podem ser definidos com a ferramenta

Tipo de input match	Descrição
In: (Text string) Out: (Move type; Strategy or task name; Value)	In: uma expressão textual. Out: pode ser gerado um <i>move</i> do tipo <i>answer</i> ou do tipo <i>answerTask</i> . Para um <i>move</i> do tipo <i>answer</i> deve ser seleccionada uma estratégia (que é do tipo <i>Findout</i>) que irá receber o valor indicado. Para um <i>move</i> do tipo <i>answerTask</i> deve ser seleccionada uma <i>cell</i> (a que corresponde uma tarefa no sentido em que a resposta do sistema consistirá em executar os planos associados a essa <i>cell</i>). Note-se que na versão actual da ferramenta cabe ao autor seleccionar as opções adequadas quando escolhe <i>answer</i> ou <i>answerTask</i> .
In: (Text string) Out: (Move type; Value)	In: uma expressão textual. Out: <i>move</i> que será passado ao sistema de diálogo. “Move type” e “value” serão aplicados sobre a “estratégia actual” (a estratégia que estiver a ser tratada no momento em que o sistema detecta a expressão de entrada). Por exemplo, se a expressão for “totalmente de acordo”, o “move type” for “answer” e o valor for “sim”, o

	sistema atribuirá o valor “sim” à estratégia que nessa altura esteja “à espera” de um valor/resposta.
In: (Text string) Out: (Move type)	In: uma expressão textual. Out: um <i>move</i> atómico (ex. <i>greet</i>). Os possíveis <i>moves</i> são listados.
In: (Text string; Variable) Out: (Move type; Strategy name; Variable)	In: uma expressão textual. Out: se no <i>input</i> do utilizador do diálogo, a seguir à “expressão textual” indicada, existir um valor válido para a estratégia indicada (a seleccionar na lista apresentada), esse valor será tomado como resposta à estratégia em questão (um <i>Findout</i>). Note-se que na versão actual da ferramenta cabe ao autor seleccionar as opções adequadas quando escolhe <i>answer</i> ou <i>answerTask</i> .

O autor selecciona da lista o tipo de *input match* a criar e escolhe “Define input match” (são suportados quatro tipos de *input match*, com a forma indicada). No exemplo da Figura 35, a uma expressão “só ida” do utilizador do diálogo, o sistema irá gerar um *move* do tipo *answer* para a estratégia “pretendeRegresso”, atribuindo-lhe o valor “não”.

Figura 35: Edição de *input matches*

Quando um novo projecto é criado são igualmente criados os *matches* genéricos, inicialmente em inglês (estes elementos são descritos no Anexo III). Estas entradas podem ser listadas seleccionando a opção “Generic matches” do menu principal.

No caso dos *input matches* não existe à partida indicação de quantos e quais os *input matches* que devem ser definidos. Aplicam-se apenas duas excepções:

- i) Deve existir pelo menos um *input match* que permita despoletar uma tarefa/plano por omissão a partir de uma indicação inicial do utilizador. Formalmente, no Midiki, o identificador de uma tarefa corresponde ao identificador de uma *cell* (relembramos que no domínio é explicitada uma tarefa por omissão, em *defaultTask*). A Figura 36 ilustra a criação de um *input match* deste tipo.

The screenshot shows a software interface titled "Input matches". At the top, there are three buttons: "BACK", "PRINT", and "DELETE". Below these are several input fields arranged in a form-like structure. The first field is "Input match position:" with the value "30". The second field is "Input expression:" with the value "reserva". The third field is "Out move type:" with a dropdown menu currently showing "answerTask". The fourth field is "Out strategy or Task:" with a dropdown menu currently showing "OrderTripCell". The fifth field is "Out value:" with the value "action:indicated task is started". At the bottom right of the form is a "Save" button.

Figura 36: *Input match* para um *answerTask*

- ii) No caso dos *matches* genéricos, estes são sempre criados com um novo projecto. O autor do diálogo não os tem de criar, mas deve verificar e editar o seu conteúdo, adequando-o ao contexto do diálogo.

Exceptuando as duas situações anteriores, os *input matches* a criar devem basear-se nas interacções esperadas entre os clientes do sistema de diálogo a criar (os seus utilizadores) e esse mesmo sistema. Isso vai depender do conhecimento que o(s) autor(es) tenha do domínio em questão e pode/deve ser igualmente determinado por testes que sejam realizados com diferentes tipos e perfis de utilizadores com o intuito de registar (anotar!) as diferentes formas de interacção esperadas e quais as expressões mais comuns/típicas.

A ordem ou posição de cada *input match* é relevante na determinação de qual a expressão que será seleccionada. Em particular quando uma expressão é uma sub-

expressão de outra maior, a expressão maior deve sempre ser inserida antes da sub-expressão, caso contrário apenas a sub-expressão será detectada.

Pelos motivos apontados sugere-se alguma ponderação na definição de *input matches*. Tratando-se de uma interacção em linguagem natural, as combinações possíveis de termos e expressões do utilizador podem ser praticamente infinitas. Em caso de dúvida sobre as situações em que pode e/ou deve ocorrer um *input match* ou havendo dúvidas sobre eventuais sobreposições com outros *input matches* é preferível não criar um *input match* a criá-lo de forma que possa prejudicar o desempenho do sistema em vez de optimizá-lo.

Definição de *slots* (6º passo da metodologia):

Ao seleccionar a opção “Cells -> Cell slots”, o autor pode criar as *slots* necessárias ao armazenamento de informação das estratégias. Um *slot* pode ser visto como um espaço de armazenamento de uma variável, embora no Midiki corresponda a um tipo não elementar (o tipo Java que serve de base aos *slots* é a *Collection*). A opção “UPDATE LIST” cria/gera uma lista de *slots* a partir das definições das estratégias dos tipos *Findout* e *Inform* existentes. A opção “NEW CELL SLOT” permite criar manualmente os *slots*, incluindo *slots* genéricos, ou seja, *slots* não directamente associados a uma determinada estratégia de um plano. Não havendo mais *slots* a definir é apresentada a mensagem “All cell slots have been defined”. A Figura 37 ilustra o menu de criação de *slots*.

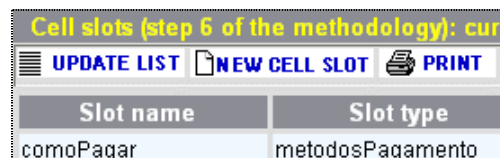


Figura 37: Menu de criação de *slots*

Declaração de *queries* do plano (7º passo da metodologia):

Ao seleccionar a opção “Cells -> Cell queries”, o autor deve confirmar os nomes das *queries* que vão implementar as funcionalidades de estratégias do tipo *QueryCall*. A opção “NEW CELL QUERY” permite especificar essas *queries*. Não havendo mais *queries* a definir é apresentada a mensagem “All cell queries have been defined”. O menu para “NEW CELL QUERY” é apresentado na Figura 38.

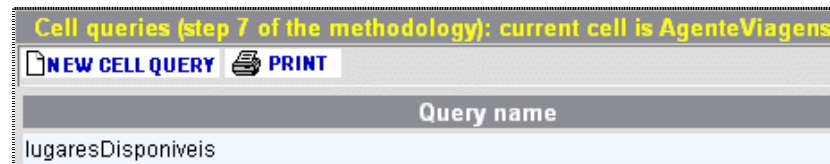


Figura 38: Menu de especificação de *cell queries*

Note-se que, mesmo que uma *query* seja invocada várias vezes nos planos do diálogo, deverá ser incluída apenas uma vez na *cell*.

Declaração de *methods* do plano (8º passo da metodologia):

A especificação de *methods* é muito semelhante à especificação das *queries*. Ao seleccionar a opção “Cells -> Cell methods”, o autor deve confirmar os nomes dos *methods* que vão implementar as funcionalidades de estratégias do tipo *MethodCall*. A opção “NEW CELL METHOD” permite especificar esses *methods*. Não havendo mais *methods* a definir é apresentada a mensagem “All cell methods have been defined”. O menu para “NEW CELL METHOD” é apresentado na Figura 39.

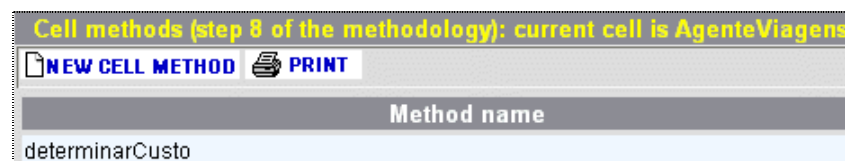


Figura 39: Menu de especificação de *cell methods*

Especificação de *handlers* (9º passo da metodologia):

A realização do passo 9 da metodologia consiste na indicação dos nomes das classes Java que implementam cada *handler*. É automaticamente indicado se se trata de um *handler* para uma *query* ou para um *method*. Não havendo mais *handlers* a definir é apresentada a mensagem “All cell handlers have been defined”. O menu para “DEFINE HANDLER” é apresentado na Figura 40.

The screenshot shows a window titled 'Cell handlers (step 9 of the methodology): current cell is AgenteViagens'. It has two buttons: 'DEFINE HANDLER' and 'PRINT'. Below these is a table with three columns: 'Query or Method name', 'Java class name', and 'Is method?'. The table contains two rows of data.

Query or Method name	Java class name	Is method?
lugaresDisponiveis	lugaresDisponiveisHandler	no
determinarCusto	determinarCustoHandler	yes

Figura 40: Menu de especificação de *handlers*

Uma vez completada a modelação do diálogo, pode ser gerado o código fonte Java que permite implementar esse diálogo no Midiki. São geradas classes de cinco tipos:

domínio, léxico, *cells*, *handlers* (os seus *templates*) e as classes das componentes ditas procedimentais, conforme passamos a descrever:

Geração automática de código para a implementação do domínio no Midiki

A opção “Generate Domain Java class” do menu principal permite obter a classe que implementa em Java o domínio do diálogo para o Midiki. Tanto neste caso como nos restantes, o código gerado é mostrado na janela do browser e deve ser gravado na pasta de domínio do projecto, na estrutura de pastas do Midiki (conforme ilustrado na Figura 56). Uma vez concluída a modelação do diálogo, geradas todas as classes na ferramenta e implementados todos os *handlers*, o código fonte deverá ser compilado juntamente com o resto do código do Midiki para criar um ambiente de execução para o diálogo.

Conforme referido na secção 3.2.3, para cada classe de domínio gerada é automaticamente incluído um plano de topo (*topPlan*), comum a qualquer projecto. Trata-se de um plano muito simples que dá as boas vindas ao utilizador (consoante a mensagem associada a “greet”) e tenta identificar qual a tarefa a executar. O controlo é depois passado para o plano principal do diálogo, no qual é executada em primeiro lugar a estratégia associada a “*Default question*”. São igualmente incluídos, de forma automática, três *moves* no fim do plano principal: i) Um *move* “thank”, que apresentará a mensagem de despedida ao utilizador; ii) Um *move* “forget”, que reinicializa todos os *slots*/estratégias, e; iii) Um *move* “addExec(top)” que reinicia o plano de topo uma vez concluída a execução do plano principal.

Geração automática de código para a implementação do léxico no Midiki

A opção “Generate Lexicon Java class” do menu principal permite obter a classe que implementa em Java o léxico do diálogo para o Midiki.

Geração automática de código para as *cells* e *handlers*

A opção “Generate Cell/Handler Java classes” do menu principal permite obter as classes Java que implementam as *cells* do diálogo e os *templates* dos *handlers*. Pode existir mais do que uma *cell* e mais do que um *handler*, pelo que o autor tem de seleccionar cada uma das *cells* e dos *handlers* e gerar separadamente cada classe. Estas funcionalidades são ilustradas na Figura 41.

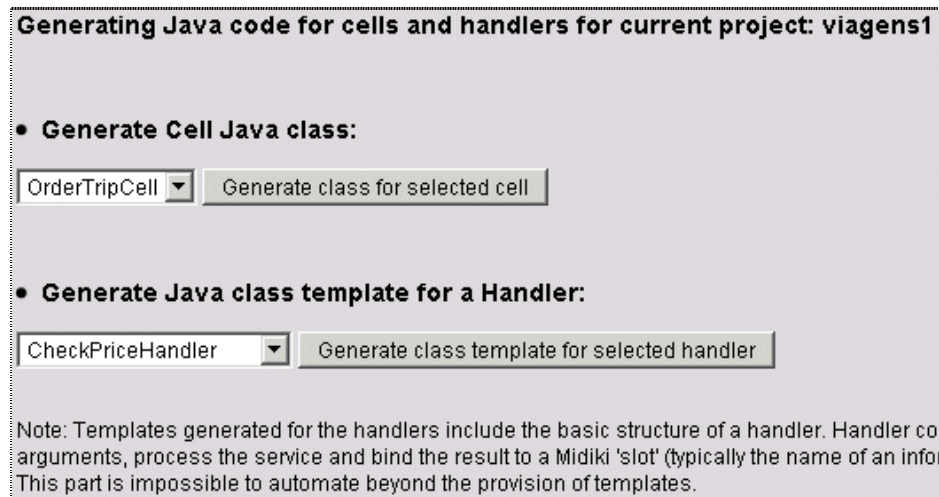


Figura 41: Ecrã de selecção de *cells* e *handlers* para geração das respectivas classes Java

Como foi referido na Secção 4.3.4, as *cells* constituem interfaces e são concretizadas através de classes do tipo *Contract*.

Geração automática do código dos componentes procedimentais

A opção “Generate Java classes procedural components” do menu principal permite obter as classes Java que implementam as componentes procedimentais do diálogo para o Midiki: classe *Tasks*, classe *DomainAgent* e classe *DomainManagerExecutive* (classe “dm”).

5.6 Opções tecnológicas

A escolha das tecnologias a empregar na implementação da ferramenta de autoria teve em conta os requisitos iniciais (apresentados na descrição geral), a disponibilidade de soluções de desenvolvimento, os ambientes para a fase de produção e ainda a nossa capacidade para utilizar eficazmente as ferramentas de desenvolvimento disponíveis. Como já foi descrito, a ferramenta de autoria foi implementada como uma aplicação *web*. As opções tecnológicas tomadas e as justificações para essas opções são as seguintes:

- O servidor aplicacional ZOPE:
 - É popular na área do desenvolvimento e disponibilização de aplicações web com base em soluções *open source* (tendo estado na origem do não menos popular gestor de conteúdos Plone).
 - O ZOPE está disponível para sistemas Windows e Linux e o código fonte de uma aplicação “ZOPE” pode ser instalado em qualquer um destes ambientes, sem modificação.

- Já possuíamos alguma experiência no desenvolvimento de aplicações web baseadas em ZOPE, os que nos dava algumas garantias de sucesso e alguma motivação acrescida.
- A utilização do SGBD MySQL¹⁸ para a implementação da base de dados foi igualmente consensual por se tratar de uma aplicação universal, disponível gratuitamente.
- O facto de querermos disponibilizar posteriormente, em regime de código aberto, de forma acessível, todo o código fonte criado (tanto em ZOPE/Python como em Java) contribui igualmente para a escolha de tecnologias disponíveis em código aberto.
- O factor custo teve igualmente algum peso nestas opções.
- O Java Emitter Templates (JET) [Popma, 2004]: À partida não tínhamos indicação de como seria feita a implementação do componente para a geração automática das classes Java para o Midiki. Nesse aspecto foi necessária alguma pesquisa até termos optado pelo JET (descrita no Anexo IV).

5.7 *Dialogue moves* e estratégias suportados pela ferramenta

Nesta secção são apresentados os tipos de *dialogue moves* e tipos de estratégias suportados pela ferramenta, os quais são descritos com detalhe no Anexo II.

Os tipos de *dialogue moves* suportados são: *ask*, *answer*, *answerTask*, *reqRep*, *greet*, *quit*, *ack*, *failed*, *topic_change* e *no_move*. A ferramenta utiliza ainda o elemento auxiliar *notDefined*. Tanto *notDefined* como *answerTask* nunca são incluídos no código Java gerado.

Os tipos de estratégias suportados são: *Findout*, *Inform*, *Assert*, *Repeat*, *reqRep*, *Ack*, *queryCall*, *methodCall*, *ifThen*, *ifThenElse*, *addMove* e *Exec*. A ferramenta utiliza ainda o elemento auxiliar *notDefined*, embora este nunca seja incluído no código Java gerado.

¹⁸ <http://www.mysql.com/>

5.8 Ferramentas utilizadas no desenvolvimento da aplicação

Apresentamos de seguida as ferramentas utilizadas no desenvolvimento e implementação da ferramenta:

- **Enterprise Architect** [Sparx, 2000-2008]: É uma ferramenta para modelação baseada na linguagem UML [OMG, 1997]. Esta ferramenta foi utilizada na modelação da base de dados e no desenho dos casos de uso;
- **Notepad++** [HO, 2005]: Esta ferramenta avançada de edição de texto para programadores foi utilizada para escrever todo o código SQL e Python, bem como para editar código HTML, Java, JET, XML e CSS;
- **Eclipse 3.2** [Eclipse, 2005]: Sendo o Eclipse um ambiente integrado de desenvolvimento com suporte para Java e outras linguagens, foi utilizado sobretudo na compilação de diálogos para o Midiki (através do Ant) e na transformação de código JET em classes Java utilizáveis. Tanto num caso como no outro, a compilação resultante é integrada num pacote JAR;
- **SQLyog** [Webyog, 2001]: Trata-se de um cliente muito completo para gestão (remota) de bases de dados mySQL, tendo sido utilizado exportar e importar a base de dados e realizar alterações à sua estrutura;
- **Microsoft Frontpage** [Microsoft, 1997-2006]: O conhecido editor HTML do Microsoft Office™ foi utilizado para definição de propriedades de apresentação dos objectos ZPT (os quais geram as páginas HTML a apresentar ao utilizador em cada momento). Propriedades tais como cores de fundo e de texto das páginas ou o layout das muitas tabelas existentes, foram ajustados recorrendo a este editor;
- **XML Copy Editor** [Schmidt, 2005-2008]: Trata-se de um editor simples e funcional para a linguagem XML. Esta ferramenta foi utilizada na escrita dos DTDs e na escrita de exemplos de código XML compatível com os DTD propostos, e;
- O próprio **ZOPE** [Fulton, 2005] oferece uma interface web a partir da qual podem ser manipulados os diferentes objectos de uma aplicação e realizadas as configurações globais (ex. tempo de *timeout* de uma sessão).

5.9 Instalação do software

A aplicação (a ferramenta) pode ser executada num servidor com seguintes componentes instalados e disponíveis:

- Servidor com sistema operativo Windows ou Linux, com conexão HTTP sobre TCP/IP activa e porta 3306 disponível para o acesso de aplicações clientes e conectores Python e Java ao MySQL;
- ZOPE-2 instalado (ex.: ZOPE 2.9.8 com interpretador Python 2.4.4)
 - *Product* ZetaDB instalado;
 - Conector ZMySQLDA para MySQL (instalado como um *Product*);
- Motor de base de dados (SGBD) MySQL >= 4.1.9;
- Java Runtime Engine >= 1.6, com o respectivo conector para MySQL (utilizamos a versão 5.1.5 - "mysql-connector-java-5.1.5-bin.jar").

A aplicação foi instalada e testada nos seguintes ambientes: "Windows XP SP2™", "Windows 2000 Server™" e Linux™ GENTOO.

O acesso à ferramenta pode ser feito a partir de qualquer computador cliente com acesso à Internet e browser com suporte para *Javascript* [Netscape, 1995] e *frames* (ex. "Internet Explorer 6™" e Firefox™). A ferramenta é invocada indicando no browser o URL da forma `http://<nome_do_servidor>:8080/Midiki2/Autoria2/menu`.

A instalação da aplicação/ferramenta segue os seguintes passos:

1. Criação da base de dados sobre MySQL: É feita executando um restore sobre o MySQL a partir do ficheiro SQL disponível, o qual cria a base de dados inicial. Esta operação pode ser efectuada por exemplo através da aplicação SQLyog;
2. Instalação do código fonte da ferramenta:
 - a. Entrando no ZOPE em modo de administração, pode ser importado o código fonte principal. O ficheiro a importar é "Midiki2.zexp", o qual deve ser previamente copiado para a pasta "import" da instância do ZOPE criada, tanto em Windows como em Linux.
 - b. As classes externas Python "myExternalPyCode.py" (que fazem a ligação ao Sistema Operativo para chamar as classes Java geradoras) são colocadas na pasta "Extensions", tanto em Windows como em Linux;
3. Instalação dos conectores: Em ambiente Windows, o conector "mysql-connector-java" deve ser colocado na sub-pasta "var" da instância do ZOPE criada. Em ambiente Linux, o conector deve ser colocado na pasta raíz da instância;
4. Instalação da biblioteca geradora: a biblioteca JAR a instalar é a biblioteca de classes geradoras (*MidikiAuthoringGenerateJava.jar*). Em Windows deve ser

colocada na sub-pasta “var” da instância do ZOPE criada. Em ambiente Linux, o conector deve ser colocado na pasta raíz da instância.

5.10 Conclusões

Neste capítulo foi apresentada a ferramenta implementada, a qual dá suporte à metodologia proposta para a modelação e prototipagem de diálogos no Midiki.

A forma como a ferramenta de autoria de diálogos foi desenvolvida e as opções de desenho que foram tomadas, nomeadamente a divisão em níveis MVC, permitem que o sistema evolua a nível da integração com diferentes sistemas operativos ou que novas funcionalidades venham a ser implementadas.

Julgamos que a ferramenta traz um contributo a esta área de investigação, em particular no que diz respeito à prototipagem de Sistemas de Diálogo, na medida em que dá suporte à metodologia proposta e automatiza grande parte da produção de código final na criação de diálogos para o Midiki.

No capítulo seguinte apresentamos um caso de estudo, implementado de raiz recorrendo sobre a ferramenta de autoria.

Capítulo 6

Caso de Estudo

6.1 Introdução

O propósito em desenvolver um caso de estudo foi utilizar a ferramenta implementada para criar um novo projecto totalmente de raiz de forma a testar e verificar a sua conformidade com os objectivos propostos, nomeadamente o de facilitar a criação de diálogos e gerar, através da ferramenta, código Java compilável com o Midiki.

Na secção 6.2 é feita a apresentação dos requisitos e casos de uso para este projecto/diálogo; Na secção 6.3 expomos a análise do problema; A secção 6.4 a apresenta os passos relativos à aplicação da metodologia a este caso, recorrendo à ferramenta de autoria; Na secção 6.5 é dada a indicação de como é feita a geração de código e a sua compilação; Na secção 6.6 é apresentada uma exemplificação de como é feita a execução do diálogo no Midiki e o capítulo termina na secção 6.7 com uma conclusão.

6.2 Requisitos e Casos de uso

A especificação do diálogo devia ser suficientemente simples (de forma a ser facilmente percebida) e, simultaneamente, devia abordar e evidenciar as principais características do Midiki e da ferramenta.

O projecto criado consistiu em simular, de forma muito simplificada, o acesso a um terminal “Multibanco”/ATM, através de interacção por fala (ou escrita), permitindo algumas das operações mais comuns disponíveis nesses terminais: i) Consultar o saldo de uma conta; ii) Efectuar um levantamento e iii) Efectuar um depósito.

Tratando-se de um caso de estudo não se pretendeu que a lista de operações suportadas fosse exaustiva ou que estas fossem reproduzidas com rigor. Os pontos mais importantes a ter em conta foram que, em termos de gestor de diálogo, fossem abordados os aspectos mais importantes da metodologia, da ferramenta de autoria e do gestor de diálogo em si (o Midiki). Os casos de uso e a lista de requisitos são ilustrados na Figura 42:

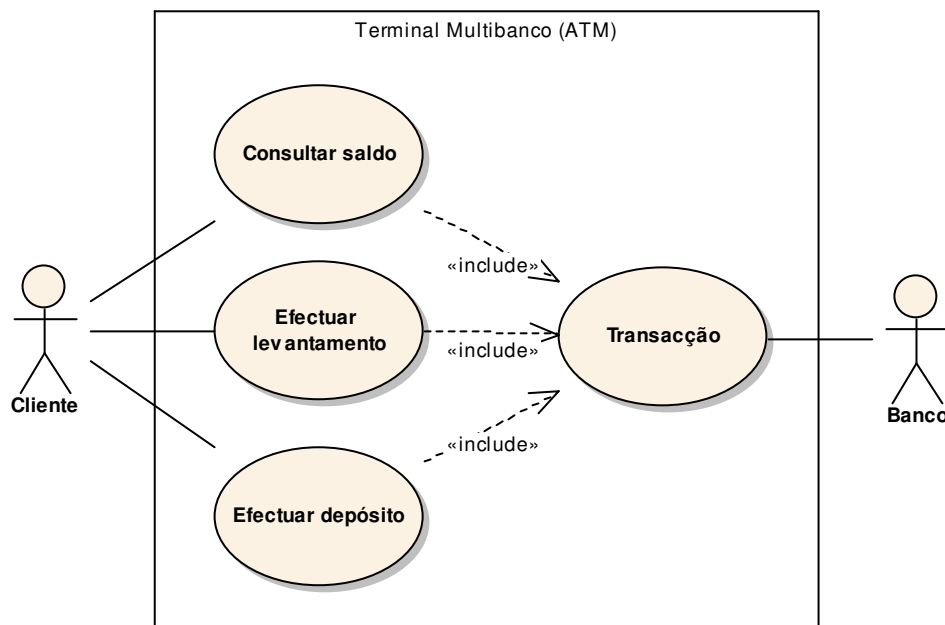


Figura 42: Casos de uso para a simulação a implementar

Actores:

Cliente: Corresponde ao utilizador do diálogo, o qual pode aceder à “máquina ATM” e realizar uma das operações definidas.

Banco: As acções do actor banco são “desempenhadas” pelo sistema de gestão de base de dados, fornecendo e guardando informação sobre os dados da conta do cliente.

Operações disponíveis / Descrição dos casos de uso:

i) **consultar o saldo** (de uma conta): Existirão 3 contas, denominadas Ordem, Prazo e Aforro. O utilizador indica qual a conta, o sistema informa qual o respectivo saldo e a operação está concluída. Estas contas são guardadas numa base de dados externa, numa tabela onde conste a identificação da conta e o respectivo saldo, actualizado em tempo real de acordo com as indicações do utilizador sobre o tipo de operação e os montantes envolvidos.

ii) **efectuar um levantamento** (de uma das três contas): O utilizador poderá fazer levantamentos de montantes entre 5 e 100 Euros (o valor a transaccionar em qualquer caso será 5 ou um múltiplo de 10). Uma vez indicada a conta, o sistema informa qual o saldo disponível e pergunta qual o valor a levantar. Se o saldo existente permitir o levantamento, é deduzido o montante no saldo, o utilizador é informado do saldo final, é-lhe dada indicação para retirar o dinheiro e a operação de levantamento de dinheiro está concluída. Se o saldo existente for insuficiente face ao levantamento pretendido, o sistema informa que não existe saldo suficiente para o valor pretendido e conclui a operação.

iii) **Efectuar um depósito** (numa das três contas): O utilizador poderá efectuar depósitos de montantes entre 5 e 100 Euros (o valor a transaccionar em qualquer caso será 5 ou um múltiplo de 10). Uma vez indicada a conta, o sistema informa qual o saldo actual e pergunta qual o valor a depositar. O utilizador indica o valor do depósito. Seguidamente o sistema informa qual o saldo final resultante e a operação está concluída.

Cada operação implicará uma transacção, concretizada por cada acesso a uma base de dados que contém informação sobre cada conta.

A interacção não se deverá limitar à sequência típica indicada devendo, sempre que possível, tirar partido das características do Midiki que permitem, por exemplo, que quando na mesma entrada (num *input* do utilizador) seja incluída informação adicional, esta possa ser tida em conta. Por exemplo, se o utilizador indicar de uma só vez que pretende efectuar um depósito e qual o montante, está a indicar duas informações relevantes: qual a operação pretendida e qual o montante a transaccionar.

Sempre que apropriado, e isso seja possível, o sistema deve dar *feedback* ao utilizador, em particular quando é detectada uma resposta claramente errada ou fora de contexto. Deverá igualmente existir uma forma de obter indicação de quais as opções permitidas em cada passo.

O utilizador deverá poder cancelar a operação, sendo que nesse caso serão assumidas as operações já realizadas. Por não ser relevante face aos objectivos pretendidos, não haverá distinção entre vários utilizadores.

6.3 Análise com vista à criação do diálogo/planos

Existe um conjunto de orientações de carácter geral que devem ser tidos em conta ao conceber-se um sistema de diálogo. A concepção de um sistema que, através de um diálogo (tipicamente por fala/voz), permita prestar um determinado serviço de forma interactiva a um utilizador, implica considerar, analisar e definir um conjunto de aspectos que devem preceder ou acompanhar a aplicação da metodologia apresentada:

- Definir os **serviços ao utilizador** a disponibilizar pelo sistema de diálogo: Implica uma análise do problema e levantamento de requisitos, de forma usual em engenharia de software. Especificar possíveis **tarefas**, identificando a lista de funcionalidades a suportar;
- Identificar as **questões** que farão parte do diálogo. São tipicamente questões que o sistema terá de colocar ao utilizador de forma a poder tratar a tarefa pretendida;
- Identificar antecipadamente possíveis **respostas** para as questões anteriores;
- Identificar e especificar os **serviços de sistema** (ou de *back end*) requeridos. Ex.: serviços que implementem regras de negócio, necessidades de acessos a uma bases de dados, etc;
- Conceber **planos** para levar a cabo a(s) tarefa(s), que permitam executar um diálogo com o utilizador: definir por que ordem o sistema irá colocar questões (e dessa forma, também qual a possível ordem das respostas do utilizador); definir/especificar as acções a tomar para além da aquisição de dados; especificar um plano principal e os subplanos; etc.

A Tabela 7 estabelece a correspondência entre este conjunto de requisitos genéricos e as diferentes partes das componentes declarativas da metodologia.

Tabela 7: Análise de requisitos genéricos e sua correspondência com os elementos da metodologia

Requisitos genéricos, conforme descrito anteriormente	Componente correspondente na metodologia e no Midiki: tipo de análise
---	---

Serviços ao utilizador (tarefas)	Não existe um componente específico, uma vez que os serviços/tarefas constituem exactamente aquilo que o sistema deve oferecer. Corresponderá a uma descrição do serviço a prestar, tendo em conta a plataforma que servirá de interface com o utilizador/utente.
Questões que o sistema terá de esclarecer/colocar ao utilizador	Domínio (planos e estratégias) e Léxico (<i>output matchers</i>): É da maior importância identificar as necessidades de informação do sistema para que este possa efectuar a tarefa pretendida. A forma mais objectiva de obter as informações necessárias é colocando questões ao utilizador, o que é concretizado através de estratégias <i>Findout</i> nos planos e respectivas saídas de texto nos <i>output matchers</i> .
Serviços de sistema a aceder	<i>Queries e Methods</i> , instanciados através de <i>handlers</i> : considera-se como um serviço uma operação ou funcionalidade externa ao GD que tem de existir para que os objectivos de uma tarefa possam ser atingidos (por exemplo uma consulta a uma tabela de uma base de dados).
Respostas possíveis do utente/utilizador	Léxico (<i>input matchers</i>): O principal interesse em antecipar possíveis respostas do utilizador é permitir identificar expressões ou termos concretos para utilização em <i>input matches</i> e em sinónimos, optimizando o sistema.
Planos necessários	Domínio (planos): Identificar e definir a ordem das interacções (típicas) entre o gestor de diálogo e o utilizador e entre o gestor de diálogo e serviços externos constitui a forma de conceber os diversos planos/subplanos necessários à criação do diálogo.

Concretizando para o presente caso de estudo os requisitos genéricos apresentados acima obtemos a análise apresentada na Tabela 8.

Tabela 8: Correspondência dos requisitos genéricos com os elementos da metodologia para o caso de estudo

Tipos de requisitos genéricos	Análise
Serviços ao utente / tarefas	Tópicos tratáveis: simulação de um acesso a uma caixa multibanco (ATM), suportando as seguintes operações: i) Consultar saldo de uma conta; ii) Efectuar levantamento; iii) Efectuar depósito.
Questões que o	Operação “consultar saldo”: Para que o sistema possa indicar ao

<p>sistema terá de esclarecer / colocar</p>	<p>cliente o saldo de uma conta necessita de obter resposta às seguintes questões:</p> <ul style="list-style-type: none"> - Qual a conta a consultar? <p>Operação “efectuar depósito”: Para que o sistema possa processar um depósito necessita de obter resposta às seguintes questões:</p> <ul style="list-style-type: none"> - Qual a conta? - Qual o montante a depositar? <p>Operação “efectuar levantamento”: Para que o sistema possa processar um levantamento necessita de obter resposta às seguintes questões:</p> <ul style="list-style-type: none"> - Qual a conta? - Qual o montante a levantar?
<p>Serviços de sistema a aceder</p>	<p>As listas de operações disponíveis, contas acessíveis e montantes permitidos por transacção serão definidas como atributos do domínio. O sistema de “back end” a aceder disponibiliza os seguintes serviços, implementados através de <i>handlers</i> Midiki:</p> <ul style="list-style-type: none"> - Obtenção do valor do saldo de uma conta; - Verificação de saldo disponível para um levantamento e actualização de saldo após o levantamento; - Actualização do saldo após um depósito; <p>(será necessário aceder a uma base de dados para manter os saldos das contas).</p>
<p>Respostas possíveis do utente / utilizador</p>	<p>Tipicamente o cliente utilizará expressões como as seguintes, em cada caso:</p> <p>Operação “consultar saldo”:</p> <ul style="list-style-type: none"> - Pretendo/gostaria de consultar o meu saldo; - Pretendo/gostaria de consultar o saldo da/na conta <conta>. <p>Operação “efectuar depósito”:</p> <ul style="list-style-type: none"> - Pretendo/gostaria de depositar <N> Euros na conta <conta>. <p>Operação “efectuar levantamento”:</p> <ul style="list-style-type: none"> - Pretendo/gostaria de levantar <N> Euros da conta <conta>. <p>Com base nestas expressões é possível definir algumas optimizações com base em <i>input matches</i>.</p> <p>Nota: <N> e <conta> serão substituídos por valores ou expressões adequadas.</p>

Planos necessários	<ul style="list-style-type: none"> - O plano principal deverá identificar qual a operação pretendida (perguntando ao utilizador) e chamar um subplano para cada uma das três operações. - Um subplano “consultar saldo” deve informar quais as contas disponíveis; perguntar qual a conta a consultar; obter o saldo da conta indicada e apresentar o valor do saldo ao cliente. - Um subplano “efectuar levantamento” deve indicar quais os montantes que podem ser transaccionados; perguntar qual a conta; informar o saldo inicial; perguntar qual o montante a levantar e informar o saldo final resultante. - Um subplano “efectuar depósito” deve indicar quais os montantes que podem ser transaccionados; perguntar qual a conta; informar o saldo inicial; perguntar qual o montante a depositar e informar o saldo final resultante.
--------------------	---

Uma vez definidos e satisfeitos os requisitos a que o sistema a implementar deve obedecer, o autor pode seguir a metodologia apresentada e utilizar a ferramenta para criar um diálogo que implemente o(s) serviço(s) pretendido(s).

6.4 Aplicação da metodologia a este caso

Começamos pela criação de um projecto que englobará e servirá de referência a todos os aspectos do diálogo. A Figura 43 ilustra a caracterização do projecto *MM_ATM*¹⁹ na ferramenta. Denominamos o domínio por *ATMDomain* e o léxico como *ATMLexicon* (esses nomes não são visíveis na Figura 43).

Project name: MM_ATM_v1

Author: Lucio

Remarks: Simular um acesso a uma caixa multibanco (ATM), suportando as seguintes quatro operações: i) Consultar saldo de uma conta; ii) Efectuar levantamento; iii) Efectuar depósito e iv) Transferir dinheiro.

Domain: [click here](#)

Lexicon: [click here](#)

Figura 43: Campos que caracterizam um projecto

¹⁹ MM_ATM é um acrónimo para “MultiModal Automatic Teller Machine”

O outro passo preliminar consiste na instanciação de uma *cell* que sirva de suporte aos planos a criar (pelo menos uma). Inicialmente apenas declaramos o nome da *cell* (a sua especificação será completada nos passos 6 a 9). A Figura 44 ilustra a caracterização base de uma *cell*.

Cell name:	ATMCell1
Description:	Cell de suporte ao diálogo para a máquina ATM. Neste caso só é necessária uma Cell
Cell slots:	click here
Cell queries:	click here
Cell methods:	click here
Cell handlers:	click here
<input type="button" value="Save"/>	

Figura 44: Campos que caracterizam uma *cell*

Uma vez instanciados o projecto e uma *cell*, podemos iniciar a modelação do diálogo, seguindo os passos da metodologia.

6.4.1 Especificação das componentes declarativas

As componentes declarativas englobam os passos 1 a 9 da metodologia, compreendendo a especificação do domínio, do léxico e das *cells*. Na secção seguinte descrevemos a concretização desses passos para o caso de estudo.

Especificação do Domínio

Os primeiros três passos da metodologia dizem respeito à especificação do domínio, descrita nas secções seguintes.

Passo 1 da metodologia - Construção de planos

Foi criado um plano principal (ATMmainPlan) e cinco subplanos, conforme se descreve a seguir.

ATMmainPlan: Plano principal, onde se identifica a operação bancária pretendida e é chamado um subplano para cada uma das três operações.

O plano começa com *Findout(qualOperacaoPretendida)* e consoante a resposta obtida é chamado um dos subplanos associados a cada uma das três operações

disponíveis (*consultarsaldoPlan*, *levantarPlan* ou *depositarPlan*). A Figura 45 ilustra a especificação de *ATMmainPlan*.

Strategies for plan 'ATMmainPlan'					
SEARCH/FILTER CLEAR CURRENT FILTER PRINT					
Position	Type	Name	Guard or SlotValue	Value if	subPlan-1
10	Findout	qualOperacaoPretendida			
13	IfThen		qualOperacaoPretendida	consultarsaldo	consultarsaldoPlan
16	IfThen		qualOperacaoPretendida	levantar	levantarPlan
19	IfThen		qualOperacaoPretendida	depositar	depositarPlan

Figura 45: Estratégias do plano principal do diálogo (*ATMmainPlan*)

consultarsaldoPlan: Este plano implementa um sub-diálogo relativo à operação “consultar saldo”. Começa por determinar qual a conta a consultar. Seguidamente o sistema consulta a base de dados e informa qual o saldo da conta indicada.

O plano começa com a estratégia *inform(contasDisponiveis)*, onde são indicadas ao cliente quais as contas existentes. De seguida é executada a estratégia *Findout(qualContaConsultar)* para ser identificada qual a conta a consultar. Depois é chamada a *query* “lerSaldoConta” que consulta a base de dados e actualiza o slot “valSaldoConsulta”. O cliente é informado do valor do saldo pretendido, em *inform(valSaldoConsulta)*. A Figura 46 ilustra a especificação de *consultarsaldoPlan*.

Strategies for plan 'consultarsaldoPlan'		
SEARCH/FILTER CLEAR CURRENT FILTER PRINT		
Position	Type	Name
8	Inform	contasDisponiveis
10	Findout	qualContaConsultar
13	QueryCall	lerSaldoConta
20	Inform	valSaldoConsulta

Figura 46: Estratégias do plano *consultarsaldoPlan*

levantarPlan: Este plano implementa um sub-diálogo relativo à operação “efectuar levantamento”. O sistema verifica e informa qual o saldo para a conta indicada e apenas permite levantamento se houver saldo disponível. Consoante exista ou não saldo suficiente, é chamado um dos subplanos *podeLevantarPlan* ou *naoPodeLevantarPlan*:

podeLevantarPlan: Plano executado quando se pretende efectuar um levantamento e o saldo permite a transacção.

naoPodeLevantarPlan: Plano executado quando se pretende efectuar um levantamento e o saldo não permite a transacção.

depositarPlan: Este plano implementa um sub-diálogo relativo à operação “efectuar depósito”. Dada a conta, o sistema informa qual o saldo actual e pergunta quanto se pretende depositar. O utilizador indica o valor do depósito. O sistema informa qual o saldo final resultante e a operação está concluída. A Figura 47 ilustra a especificação de *depositarPlan*, listando as estratégias incluídas nesse plano. Na Figura 47, *valSaldoDeposito1* diz respeito ao saldo antes do depósito e *valSaldoDeposito2* diz respeito ao saldo depois do depósito.

Strategies for plan 'depositarPlan'		
<input type="text"/> SEARCH/FILTER <input checked="" type="checkbox"/> CLEAR CURRENT FILTER <input type="button" value="PRINT"/>		
Position	Type	Name
8	Inform	montantesPodeDepositar
10	Findout	qualContaDepositar
12	QueryCall	lerSaldoConta
16	Inform	valSaldoDeposito1
19	Findout	montanteDepositar
25	QueryCall	actualizarSaldoDepositar
28	Inform	valSaldoDeposito2

Figura 47: Estratégias do plano *depositarPlan*

Note-se que o campo “Position” indica a ordem inicial de execução (ou inicialização) de cada estratégia. Este campo pode ser alterado, mas deve ser mantida a ordem lógica de execução das estratégias no plano, de forma que a sua sequência faça sentido para a tarefa em questão.

Passo 2 da metodologia - Inicialização de atributos e de sinónimos

Para a implementação do serviço do tipo ATM descrito foram considerados os seguintes atributos:

operacoes: A lista de operações suportadas é {consultarsaldo, levantar, depositar}.

contas: Os tipos de contas disponíveis são {ordem, prazo e aforro}.

notas: Poderão transaccionar-se valores entre 5 e 100 Euros, em múltiplos de 10, pelo que os valores para “notas” é o conjunto {5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}.

simnao: Este tipo é composto pelos valores {sim, não} e serve para limitar de forma genérica respostas do tipo sim/não (ex. para um utilizador confirmar se pretende de facto uma determinada opção, a sua resposta será deste tipo).

Para todos estes valores de atributos foram definidos sinónimos. Como exemplo, para o valor “consultarsaldo”, do atributo “operacoes”, foram definidos os seguintes

sinónimos: {consultar, consulta, conferir, verificar, obter saldo, ver saldo}, o que significa que qualquer um destes termos será uma resposta válida para *Findout(qualOperacaoPretendida)* no plano principal do diálogo. Se o utilizador responder “verificar saldo”, o valor guardado no *slot* de *qualOperacaoPretendida* será “consultarsaldo”, o mesmo acontecendo se for referido qualquer outros dos seus sinónimos.

A Figura 48 ilustra o caso do atributo “contas”, para o qual foram definidos os valores {ordem, prazo e aforro} (os sinónimos não são visíveis na Figura 48).





Attributes and synonyms (step 2 of the methodology): current project is MM_ATM_v1	
 NEW ATTRIBUTE	 PRINT
Attribute type	
contas	As contas são: ordem, prazo e aforro
Attributes and synonyms (step 2 of the methodology): current project is MM_ATM_v1	
 NEW ATTRIBUTE VALUE	 PRINT
Value	Synonyms
ordem	click here
prazo	click here
aforro	click here

Figura 48: Definição do atributo “contas” e seus valores

Como já foi referido na secção 5.5 (no 2º passo da metodologia), o atributo *generalAttributeForSynonyms* é automaticamente criado com um novo projecto e tem como objectivo criar listas de sinónimos para termos utilizados em *input matches*. Por exemplo, o termo “perdao” está associado a um *input match* genérico da seguinte forma: *addInputTokens("perdao"); addOutputString("reqRep")*. Isto significa que se num *input* o utilizador incluir o termo “perdao”, o sistema responderá com um *dialogue move* do tipo *reqRep* (que significa “request repeat”), repetindo a questão anterior. No entanto o utilizador pode utilizar outras expressões ou termos com o mesmo significado de “perdao”. Neste caso definimos os seguintes sinónimos para “perdao”: {perdão, nao percebi, não percebi, o que foi que disse, o que foi, perdoe-me, o quê, repita}.

Passo 3 da metodologia - Inicialização de questões

Uma vez criados todos os planos e estratégias, devem ser indicados quais os tipos de valores para as estratégias *Findout*. A importância deste passo reside em possibilitar que todas as questões que possam ter como resposta um valor limitado a um conjunto de valores sejam mais facilmente tratadas pelo sistema, uma vez que é possível

verificar a validade de cada resposta. A Figura 49 ilustra os tipos de questões existentes neste projecto.

Questiontypes (step 3 of the methodology): current project is MM_ATM_v1

NEW QUESTION TYPE PRINT

Strategy/Question	Attribute
qualOperacaoPretendida	operacoes
qualContaConsultar	contas
qualContaLevantar	contas
montanteLevantar	notas
qualContaDepositar	contas
montanteDepositar	notas

Figura 49: Tipos de questões do diálogo

Especificação do Léxico

Os passos 4 e 5 da metodologia dizem respeito à especificação do léxico. A aplicação desses dois passos para o caso de estudo é descrita nas secções seguintes.

Passo 4 da metodologia – Correspondência de *output matches*

Para cada *Findout* e para cada *Inform* dos planos do diálogo, indicamos qual o texto a apresentar ao utilizador/cliente do serviço ATM. Como exemplos, consideremos *Findout* (*qualOperacaoPretendida*) e *Inform* (*valSaldoConsulta*), apresentados nas Figuras 50 e 51, respectivamente:

Output matches

BACK PRINT DELETE

Output match position: 13

Strategy: qualOperacaoPretendida

Main output text: Indique a operação pretendida.

Text to output if user answer error: Desculpe, não percebi qual a operação pretendida.

Save

Figura 50: Definição do *output match* para *Findout*(*qualOperacaoPretendida*)

Figura 51: Definição do *output match* para *Inform(valSaldoConsulta)*

Os *output matches* genéricos foram editados no sentido de adaptá-los ao contexto do projecto/diálogo em questão. Por exemplo, para um “greet” do sistema para o utilizador, a mensagem a apresentar é “Bem vindo ao novo terminal multibanco com interacção multimodal. Pode utilizar este serviço para aceder a uma das suas contas e consultar o saldo, levantar ou depositar dinheiro.”.

Ao contrário do que acontece com as estratégias nos planos e com os *input matches*, no caso dos *output matches* não é particularmente relevante a sua posição (campo “position”), uma vez que cada *output match* é univocamente identificado. O número de *output matches* a criar depende apenas da quantidade de estratégias *Findout* e *Inform* existentes nos planos do diálogo.

Passo 5 da metodologia - Correspondência de *input matches*

No caso dos *input matches* não existe à partida indicação de quantos e quais os *input matches* que devem ser definidos. Relativamente aos *input matches* definidos especificamente para este sistema, apresentamos dois exemplos na Tabela 9:

Tabela 9: *Input matches* definidos para o “serviço ATM”

Expressão de entrada	Dialogue move a gerar	Estratégia	Valores a atribuir a uma estratégia ou Uma acção a realizar
“à ordem”	answer	notDefined	ordem
Este caso corresponde a um <i>input match</i> da forma “In: (Text string) -> Out: (Move type; Value)”. Desde que a expressão de entrada dada pelo utilizador contenha “à ordem” o sistema			

assumirá “ordem” como uma resposta à estratégia “actual”, independentemente do seu nome, posição ou plano.

“a odem”	answer	notDefined	ordem
----------	--------	------------	-------

Este *input match* foi definido tendo em conta que, na interacção por fala, a expressão proferida como “à ordem” é recorrentemente reconhecida como “a odem”. Esta situação pode ser corrigida no próprio software de reconhecimento, mas também é possível fazê-lo no gestor de diálogo, sendo claro que a expressão errada “a odem” deriva sempre de “à ordem”. Em vez do sistema gerar uma mensagem de “erro” (repetindo a questão) ao receber a expressão “a odem”, interpreta essa expressão como significando “à ordem”.

Relativamente a um *input match* que corresponda a um *move* do tipo *answerTask* que possa despoletar a tarefa “ATMCell1”, este é ilustrado na Figura 52. Caso o utilizador do diálogo refira o termo “pretendo” (ou um sinónimo desse termo), é executado um *move* da forma *answer(task, “ATMCell1”)*. Deve existir um *input match* que corresponda a um *answerTask*. Optamos por incluí-lo na última posição (199) dado que só é executado uma vez e não se confunde com outros *input matches*.

The screenshot shows a software interface for configuring input matches. The title is "Input matches". At the top, there are three buttons: "BACK", "PRINT", and "DELETE". Below these are several input fields and dropdown menus:

- Input match position:** A text field containing the number "199".
- Input token:** A text field containing the word "pretendo".
- Out move type:** A dropdown menu with "answerTask" selected.
- Out strategy or Task:** A dropdown menu with "ATMCell1" selected.
- Out value:** A text field containing the string "action:indicated task is started".

At the bottom right of the form is a "Save" button.

Figura 52: *Input match* que corresponde a um *answerTask*

Relativamente aos *input matches* genéricos, estes foram editados para incluir os termos considerados mais apropriados. Por exemplo, a expressão de base que indica ao sistema um “greet” do utilizador é “ola”, que por sua vez possui os seguintes sinónimos: {olá, boa tarde, boa noite, bom dia, viva}.

Especificação das Cells

Os passos 6 a 9 da metodologia dizem respeito à especificação dos elementos que compõem uma *cell*. A aplicação destes passos para o caso de estudo é descrita nas secções seguintes.

Passo 6 da metodologia – Definição/declaração de *slots*

Foram criados os *slots* indicados na Figura 53. Podemos reconhecer claramente entre os *slots* que possuem um tipo bem definido (indicado em “Slot type”) e os que possuem o tipo “notDefined”. Os *slots* que correspondem a estratégias *Findout* possuem um tipo bem definido, o que permite ao sistema validar as respostas dadas pelo utilizador. Os *slots* com tipo “notDefined” correspondem a valores indefinidos (ou genéricos). No presente caso todos os *slots* com tipo “notDefined” destinam-se a valores numéricos não enumeráveis (são utilizados por estratégias *Inform* para apresentar saldos das diversas contas em diversas operações). O *slot* “existeSaldo” corresponde a um *slot* “genérico” pois não está associado a um *Findout* ou a um *Inform*. O valor deste *slot* é definido no *handler* de *QueryCall(verificarSaldoOk)*, sendo posteriormente acedido por *IfThenElse(existeSaldo, sim, podeLevantarPlan, naoPodeLevantarPlan)*, no plano. O *slot* *existeSaldo* toma valores em {sim, nao}.

Cell strategy types (step 6 of the methodology): current cell is ATMCell1

UPDATE LIST NEW CELL SLOT PRINT

Slot name	Slot type
valSaldoConsulta	notDefined
valSaldoDeposito1	notDefined
valSaldoDeposito2	notDefined
valSaldoLevantar1	notDefined
valSaldoLevantar2	notDefined
qualOperacaoPretendida	operacoes
qualContaConsultar	contas
qualContaLevantar	contas
montanteLevantar	notas
qualContaDepositar	contas
montanteDepositar	notas
existeSaldo	simnao

Figura 53: *Slots* criados para o diálogo do caso de estudo

Passo 7 da metodologia - Declaração das queries implementadas

As *queries* existentes são *lerSaldoConta*, *actualizarSaldoDepositar* e *verificarSaldoOk*.

lerSaldoConta: Esta *query* é chamada nos planos *consultarsaldoPlan*, *levantarPlan* e *depositarPlan* e de acordo com a operação actual, devolve resultados em diferentes *slots*/estratégias desses planos e ainda em “existeSaldo”.

actualizarSaldoDepositar: Esta *query* pertence ao plano *depositarPlan* e faz actualização do saldo nos depósitos.

verificarSaldoOk: Esta *query* pertence ao plano *levantarPlan* e verifica se o cliente pode levantar o montante pretendido (ter em conta que a *query* é processada no seu *handler*). Caso a resposta seja afirmativa o valor é logo deduzido do saldo e é actualizado o *slot* relativo ao resultado (estratégia *valSaldoLevantar2*, no plano *podeLevantarPlan*).

Passo 8 da metodologia - Declaração dos methods implementados

Não foram criados *methods*, mas a sua declaração e processamento são idênticos às *queries*.

Passo 9 da metodologia – Inclusão de referências dos handlers às respectivas classes Java

Os *handlers* das três *queries* definidas são os indicados na Figura 54. Essas classes devem existir na pasta do projecto (a pasta do projecto do caso de estudo chama-se “ATMDomain” e é mostrada na Figura 56).



Cell handlers (step 9 of the methodology): current cell is ATMCell1		
 DEFINE HANDLER  PRINT		
Query or Method name	Java class name	Is method?
lerSaldoConta	lerSaldoContaHandler	no
actualizarSaldoDepositar	actualizarSaldoDepositarHandler	no
verificarSaldoOk	verificarSaldoOkHandler	no

Figura 54: *Handlers* para as *queries* existentes

6.4.2 Criação das componentes procedimentais

As componentes procedimentais englobam os passos 10 a 13 da metodologia. Nesta secção fazemos uma breve descrição do código implementado para os vários *handlers* (passo 10) e do código gerado pela ferramenta para os passos 11 a 13 da metodologia.

Passo 10 da metodologia – Implementação de handlers

Handler *lerSaldoContaHandler*: implementa o tratamento da query *lerSaldoConta*, a qual é chamada nos planos *consultarsaldoPlan*, *levantarPlan* e *depositarPlan*. Consoante o caso, é feito um tratamento apropriado, seguindo o seguinte algoritmo:

- É identificada qual a operação em curso por consulta ao *slot* “qualOperacaoPretendida”
- Se operação = “consultarsaldo”, obter nome da conta em “qualContaConsultar”
- Se operação = “levantar”, obter nome da conta em “qualContaLevantar ”
- Se operação = “ depositar”, obter nome da conta em “qualContaDepositar”
- É aberta uma ligação à base de dados e obtido o saldo para a conta indicada
- Se operação = “consultarsaldo”, devolver o resultado/saldo em “valSaldoConsulta”
- Se operação = “levantar”, devolver resultado/saldo em “valSaldoLevantar1”
- Se operação = “ depositar”, devolver resultado/saldo em “valSaldoDeposito1”

Handler *actualizarSaldoDepositarHandler*: implementa o tratamento da query *actualizarSaldoDepositar*, a qual é chamada no plano *depositarPlan*. Este handler possui o seguinte algoritmo:

- É identificada a conta para o depósito por consulta ao *slot* “qualContaDepositar”
- É obtido o montante a depositar por consulta ao *slot* “montanteDepositar”
- É aberta uma ligação à base de dados e actualizado o saldo para a conta indicada
- O saldo final é gravado/devolvido no *slot* “valSaldoDeposito2”

Handler *verificarSaldoOkHandler*: implementa o tratamento da query *verificarSaldoOk*, a qual é chamada no plano *levantarPlan*. Este handler possui o seguinte algoritmo:

- É identificada a conta para o levantamento por consulta ao *slot* “qualContaLevantar”
- É obtido o montante a depositar por consulta ao *slot* “montanteLevantar”
- É aberta uma ligação à base de dados e feita a verificação de suficiência de saldo para levantamento na conta indicada
- Em caso afirmativo, efectuar duas operações:
 - Definir “existeSaldo” com valor “sim”
 - Indicar saldo final no *slot* “valSaldoDeposito2”
- Caso o saldo existente seja insuficiente para o levantamento pretendido, definir “existeSaldo” com valor “nao”

Passo 11 da metodologia – Parametrização da classe Tasks

Esta classe inclui referência ao domínio (*ATMDomain*) e às *cells* criadas (*ATMCell1*). Esta parametrização é realizada pela ferramenta e a classe pode ser obtida de forma automática.

Passo 12 da metodologia – Parametrização da classe *DomainAgent*

Esta classe inclui igualmente referência ao domínio (*ATMDomain*) e às *cells* criadas (*ATMCell1*). Esta parametrização é realizada pela ferramenta e a classe pode ser obtida de forma automática.

Passo 13 da metodologia – Parametrização da classe “dm”

A classe “dm” (*dialogue manager executive*) foi gravada com o nome “ATMDomain_dm.java”. Esta classe inclui referência ao domínio (*ATMDomain*). Esta parametrização é realizada pela ferramenta e a classe pode ser obtida de forma automática.

É a classe de “arranque” do diálogo quando executado após compilação no Midiki. É feita a verificação da eventual inclusão do argumento “voice” na sua invocação, para suporte à integração com o Philips Freespeech 2000 [Guilhoto & Rosa, 2001] quando se pretende realizar interacção também por fala (com a configuração apresentada na Figura 58).

É também nesta classe que se pode incluir uma das opções de *debugging* durante a execução de um diálogo: o “Domain Spy”, o qual é incluído por omissão, mas que pode ser desactivado removendo a linha correspondente nesta classe, a qual se encontra devidamente identificada com “add the following line to activate Domain Spy” no código gerado para a classe.

6.5 Geração de código e compilação

As nove classes que compõem este projecto/diálogo são *ATMDomain.java*; *ATMLexicon.java*; *ATMCell1.java*; *Tasks.java*; *DomainAgent.java*; *verificarSaldoOkHandler.java*; *lerSaldoContaHandler.java*; *actualizarSaldoDepositarHandler.java* e *ATMDomain_dm.java*. Estas classes foram obtidas conforme acabamos de descrever nas secções anteriores.

Compilação das fontes Java para o Midiki

Antes de podermos compilar um novo diálogo será necessário importar a estrutura de classes original do Midiki para um ambiente de desenvolvimento Java (basta realizar esta tarefa uma vez). No Eclipse, a estrutura resultante é a ilustrada pela Figura 55.

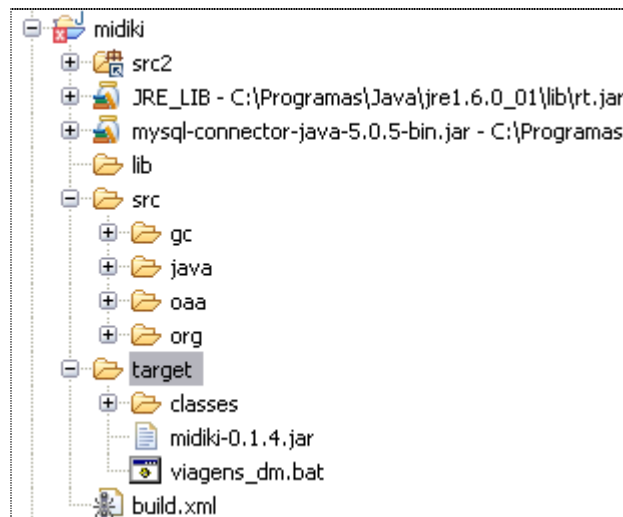


Figura 55: Estrutura de classes do Midiki integradas no *workspace* do Eclipse

O código fonte do Midiki compõe-se de cerca de 200 classes Java e a sua estrutura, tal como é reproduzida na Figura 56, foi já apresentada e descrita no capítulo 3.

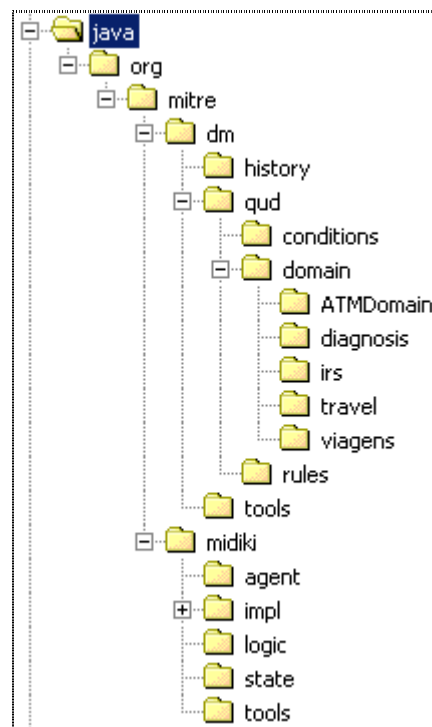


Figura 56: Detalhe da estrutura de pastas do Midiki (pasta "java")

As classes em código fonte Java geradas para um novo diálogo deverão ser gravadas numa subpasta da pasta geral “domain”, a qual possui o caminho “src\java\org\mitre\dm\qud\domain”, no *workspace* do Midiki no Eclipse. Essa subpasta terá o mesmo nome do domínio do diálogo criado. No caso presente, a

subpasta chama-se “ATMDomain”. Na Figura 56 podemos constatar que existem cinco pastas de diálogos no *workspace* (ATMDomain, diagnosis, irs, travel e viagens).

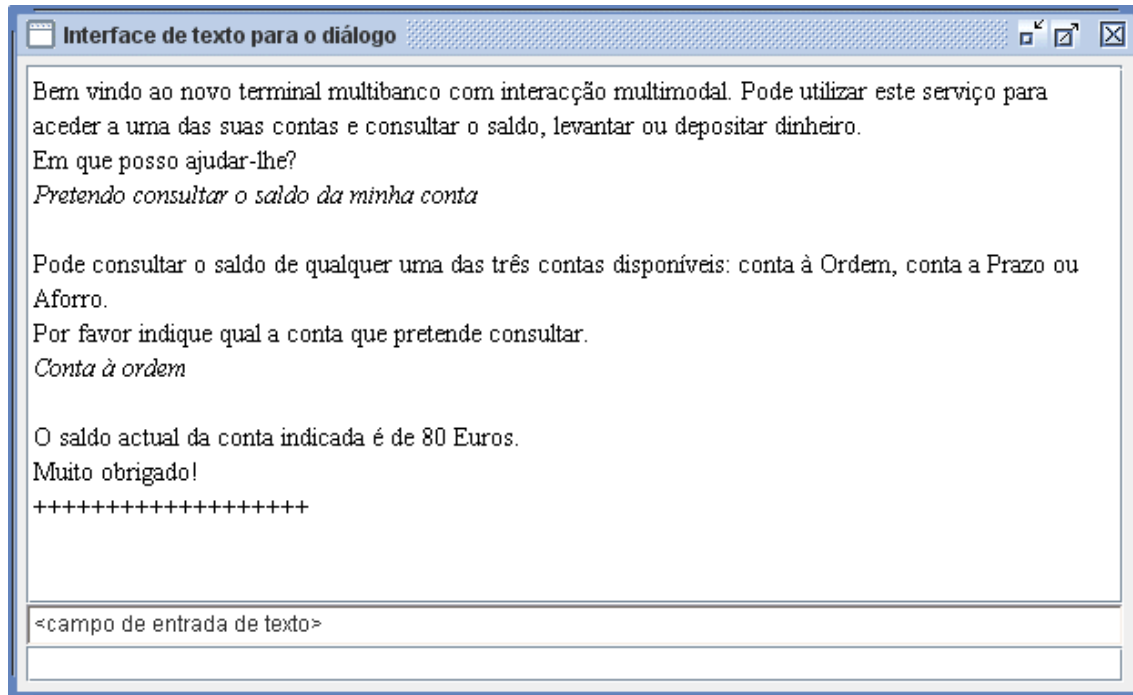
A compilação é realizada executando o “comando” *build.xml* através da ferramenta “Ant Build”, integrada no Eclipse. O pacote JAR resultante, gravado na pasta “target” do *workspace* corresponde a um ficheiro com o nome “midiki-0.1.4.jar”. Este JAR contém todas as classes base do Midiki mais as classes adicionadas para o novo diálogo. Passamos a descrever como deve ser feita a invocação das classes que executam um diálogo.

6.6 Execução do diálogo no Midiki

Uma vez compilado um novo diálogo, este pode ser testado numa interacção com base em texto escrito, conforme ilustrado na Figura 57. Como exemplificação do potencial da utilização dos diálogos criados recorrendo a interacções por reconhecimento e síntese de fala, apresentamos uma configuração testada, baseada no Philips FreeSpeech 2000 [Guilhoto & Rosa, 2001], para o reconhecimento de fala, e no *Microsoft Speech API* (MSAPI, versão 5) [Microsoft, 2007], para a síntese de fala via Text-To-Speech - TTS.

6.6.1 Correr o diálogo numa janela de texto

Para o exemplo da “máquina ATM”, o comando a invocar numa linha de comandos Windows que disponibilize o comando “java” é: `>java -classpath mysql-connector-java-5.0.5-bin.jar;midiki-0.1.4.jar org.mitre.dm.qud.domain.ATMDomain.ATMDomain_dm no_voice`. Os dois pacotes JAR indicados devem coexistir na mesma pasta. Opcionalmente pode ser passado um argumento “voice”/“no_voice” que indica que se pretende utilizar, ou não, saída de texto para voz/fala (via TTS/MSAPI). Se não for indicado um argumento o comando não funciona.



**Figura 57: Janela de interacção por texto em linguagem natural
(sessão nº 1)**

A sessão nº 1, apresentada na Figura 57, corresponde a uma interacção típica, com uma sequência de “execução” das estratégias coincidente com a sua sequência nos planos do diálogo. O valor indicado para o saldo (80 Euros) provém da consulta à base de dados.

Seguidamente transcrevemos mais três sessões realizadas com a nossa “máquina ATM”, onde podemos evidenciar algumas das potencialidades do Midiki e do diálogo implementado (“GD>” corresponde ao Gestor de Diálogo e “U>” corresponde ao utilizador humano que acede ao serviço):

Sessão nº 2:

GD> Bem vindo ao novo terminal multibanco com interacção multimodal. Pode utilizar este serviço para aceder a uma das suas contas e consultar o saldo, levantar ou depositar dinheiro. Em que posso ajudar-lhe?

U> *Quero depositar 5 Euros na conta à ordem*

GD> Pode efectuar depósitos de valores entre 5 e 100 Euros, em múltiplos de 10. O saldo actual da conta indicada é de 65 Euros. O saldo final da sua conta, após o depósito, é de 70 Euros.

----- Muito obrigado-----

Na 2ª sessão o utilizador indica logo de início três respostas para três questões pendentes nos planos do diálogo: 1) Qual a operação pretendida? 2) Qual a conta a aceder? e 3) Qual o montante a transaccionar?.

De “uma só vez” o sistema obtém a informação necessária e dá a resposta pretendida pelo utilizador. A capacidade do sistema em interpretar devidamente uma entrada como “*Quero depositar 5 Euros na conta à ordem*” depende não apenas do próprio gestor de diálogo em si, mas também da capacidade do autor em criar mecanismos de “*input match*” que permitam interceptar determinadas expressões e delas obter a informação desejada, de forma correcta. Como podemos constatar, pelo diálogo da sessão nº2, já não foram colocadas as questões para *Findout(qualContaDepositar)* e *Findout(montanteDepositar)* do plano 'depositarPlan', uma vez que as respectivas respostas já se encontravam disponíveis. Dependendo da forma como o utilizador do diálogo se exprime, da forma como o diálogo é implementado e do próprio gestor de diálogo, o sistema pode ou não ser capaz de detectar e utilizar a informação adicional “disponibilizada”. Sempre que haja informação em falta para estratégias de um plano serão colocadas as questões correspondentes a essas estratégias.

Sessão nº 3:

GD>Bem vindo ao novo terminal multibanco com interacção multimodal. Pode utilizar este serviço para aceder a uma das suas contas e consultar o saldo, levantar ou depositar dinheiro. Em que posso ajudar-lhe?

U>*Gostaria de efectuar um levantamento*

GD>Pode efectuar levantamentos de valores entre 5 e 100 Euros, em múltiplos de 10. De que conta pretende efectuar o levantamento?

U>*Quais as opções*

GD>As opções são: ordem, prazo e aforro. De que conta pretende efectuar o levantamento?

U>*Da conta aforro*

GD>O saldo actual da conta indicada é de 105 Euros. Quanto pretende levantar?

U>*25 Euros*

GD> Por favor repita qual a quantia a levantar. Pode efectuar levantamentos de valores entre 5 e 100 Euros, em múltiplos de 10. Quanto pretende levantar?

U>20 Euros

GD>O saldo final da conta após o levantamento é de 85 Euros. Por favor retire o seu dinheiro.

----- Muito obrigado-----

Na 3ª sessão o utilizador começa por indicar a operação pretendida, mas pede indicação de quais as contas disponíveis ao perguntar “quais as opções”. Ao indicar “25 Euros” como quantia a levantar, o utilizador indicou um valor não válido, pelo que o sistema pede novamente a indicação de qual a quantia a levantar, indicando quais os levantamentos possíveis. A sessão termina com a indicação do saldo final resultante, a indicação ao cliente para “retirar o seu dinheiro” e um agradecimento.

Sessão nº 4:

GD>Bem vindo ao novo terminal multibanco com interacção multimodal. Pode utilizar este serviço para aceder a uma das suas contas e consultar o saldo, levantar ou depositar dinheiro. Em que posso ajudar-lhe?

U>*Quero depositar 50 euros*

GD>Pode efectuar depósitos de valores entre 5 e 100 Euros, em múltiplos de 10. Em que conta pretende efectuar o depósito?

U>*Conta a termo*

GD>O saldo actual da conta indicada é de 65 Euros. O saldo final da sua conta, após o depósito, é de 115 Euros.

----- Muito obrigado-----

Na 4ª sessão o utilizador dá logo indicação de qual a operação pretendida e qual o montante envolvido (duas respostas num mesmo *input*). O sistema pergunta qual a conta e, após efectuar a transacção correspondente ao depósito, indica o saldo final. Neste caso o utilizador referiu um termo alternativo para “prazo”, tendo indicado que pretendia efectuar um depósito na conta a “termo”. A palavra “termo” é um dos sinónimos de “prazo” definidos.

Estes quatro exemplos anteriores demonstram a flexibilidade que o gestor de diálogo permite na interacção em linguagem natural com um utilizador humano. Seguidamente apresentamos a configuração criada para uma interacção por fala.

6.6.2 Um exemplo de interacção por fala com reconhecimento e síntese de fala em Português europeu

Recorrendo a aplicações *standard* podemos testar o nosso diálogo através da interacção por fala. Neste exemplo, sobre a janela base (um objecto JPanel) para interacção baseada em texto escrito, procedemos à colocação de texto proveniente do reconhecimento de fala do Philips FreeSpeech 2000 e procedemos ao output “falado” por TTS/MSAPI, o qual é controlado pela aplicação DSpeech [Coutsoumbas, 2007], realizando a função “Text to Speech – TTS” através da activação da opção “Get clipboard” do DSpeech. Esta configuração é compatível com qualquer diálogo criado. O diagrama que exemplifica como funciona esta integração é ilustrado pela Figura 58.

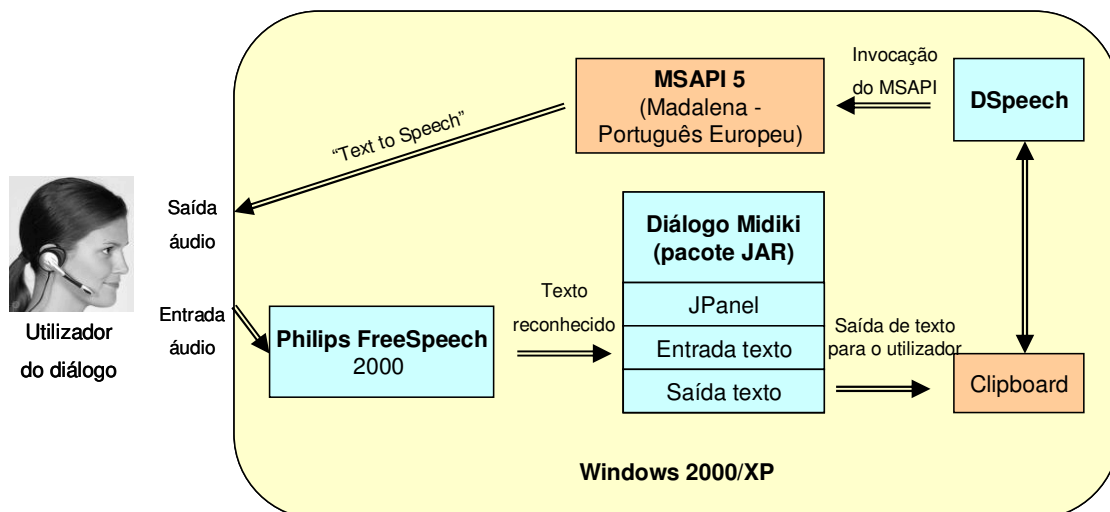


Figura 58: Uma configuração para interacção por fala com os diálogos criados

Na Figura 59 podemos ver a barra de opções do Philips FreeSpeech 2000 (modo “ditado” activo).



Figura 59: Barra de opções do Philips FreeSpeech 2000

No presente caso o comando a invocar numa linha de comandos Windows com Java seria:


```
">java -classpath mysql-connector-java-5.0.5-bin.jar;midiki-0.1.4.jar  
org.mitre.dm.qud.domain.ATMDomain.ATMDomain_dm voice"
```

O sistema apresentado na Figura 58 funciona localmente, com todas as aplicações instaladas na mesma máquina. Neste caso janela de interacção por texto mantém-se e o texto de entrada pode ser também digitado ou então incluído automaticamente pelo FreeSpeech no campo “Entrada de texto” (o texto reconhecido é inserido na posição actual do cursor).

Relativamente à qualidade do reconhecimento de fala no Philips FreeSpeech 2000, é importante referir que esta ferramenta requer um treino específico para cada utilizador. Nos nossos testes utilizamos um sistema com apenas 1 hora e 30 minutos de treino gravado para Português europeu (um utilizador), e os resultados foram muito aceitáveis.

6.6.3 Debugging de um diálogo em runtime

A compilação de um diálogo com as opções definidas por omissão faz com que sejam exibidas duas janelas adicionais quando executamos o diálogo. Essas duas janelas adicionais possuem os nomes “Domain Spy” e “Information State Viewer”. O “Domain Spy” apresenta a estrutura dos planos em memória para um determinado diálogo. Por sua vez o “Information State Viewer” permite acompanhar o estado do *information state* em cada passo, registando cada actualização feita ao longo da sessão do diálogo.

6.7 Conclusão

Neste capítulo apresentámos como caso de estudo foi implementado, foi explicado como deve ser compilado o diálogo e como este pode ser executado. Foi apresentada uma configuração que demonstra a possibilidade de interacção baseada apenas em texto, apenas em fala ou em texto e fala. No final do capítulo foram descritas as opções de *debugging* de um diálogo em *runtime*.

Não avaliamos formalmente os ganhos decorrentes de se utilizar a ferramenta versus trabalhar apenas com classes Java (ex.: facilidade de implementação, eficiência acrescida, amigabilidade do interface, maior correcção do código, etc), contudo podemos constatar que isso apresenta várias vantagens, designadamente:

- A ferramenta tem como elemento central o “projecto”, o qual engloba todas as componentes da metodologia, as quais são claramente identificadas e tratadas através da interface disponibilizada. A vista que se tem sobre um projecto

apenas apresenta os aspectos relevantes para o autor, escondendo detalhes de codificação/implementação;

- O autor não precisa trabalhar sobre classes Java, mas apenas com a informação relevante (campos) que deve caracterizar cada objecto. Operações de edição consistem em criar, editar e eliminar objectos e não em seleccionar e editar secções de um documento Java;
- O projecto é mantido numa base de dados e a interface *web* da ferramenta permite que se possa trabalhar num projecto a partir de qualquer local com acesso ao servidor;
- O código gerado reflecte sempre as alterações mais recentes, sem necessidade de se ter de ter em conta repercussões de alterações de uma classe sobre outras classes, e;
- Praticamente todo o código necessário a um projecto/diálogo pode ser gerado através da ferramenta. Isso permite garantir uma maior correcção e consistência do código final produzido (Java).

Reconhecemos que a implementação de um sistema deste tipo que deva operar num ambiente real apresenta requisitos e desafios que vão além do caso aqui apresentado. Não obstante, seria um desafio que certamente aceitaríamos com confiança. Nesse cenário existiriam novos requisitos a ter em conta, nomeadamente os seguintes: Que tipo de interface a criar (apenas texto?; apenas fala? texto+fala?); Funcionamento local vs funcionamento distribuído; Independência e qualidade do reconhecimento de fala do utilizador no caso da interacção por fala; Equipamento terminal (ex. quiosque, PDA, PC, telefone).

Capítulo 7

Conclusões

Nesta dissertação investigamos a temática da interacção Pessoa-Máquina no que diz respeito à criação de Sistemas de Diálogo (SD) e à criação de Gestores de Diálogo (GD), incluindo metodologias, abordagens e ferramentas utilizadas para a modelação de diálogos. De entre as abordagens analisadas, demos maior ênfase à *Information State Update* (ISU), sendo essa a abordagem utilizada pelo GD que adoptamos (o Midiki). A existência de poucas metodologias e ferramentas de autoria que possibilitem uma modelação fácil e intuitiva de novos diálogos motivou-nos para a proposta de uma metodologia e uma ferramenta de autoria para a “criação” de diálogos com base no Midiki.

As duas contribuições mais importantes deste trabalho são, assim:

- i) A apresentação de uma metodologia para a criação/modelação de diálogos para o Midiki, e;
- ii) A implementação de uma ferramenta, que facilita em muitos aspectos a aplicação da metodologia na criação de diálogos, automatizando consideravelmente todo o processo de geração de código final, em linguagem Java.

Estas contribuições ajudam a colmatar a falta de metodologias e de ferramentas para a modelação e prototipagem de diálogos.

De uma forma geral foi uma experiência gratificante e muito enriquecedora, revelando-se, ao mesmo tempo, bastante exigente. Destacamos as duas seguintes “lições”:

- A aprendizagem colateral foi ampla. Houve necessidade de abordarmos e lidarmos com uma grande variedade de assuntos e tecnologias que, dando suporte à solução do problema central, tiveram de ser aprendidas e aplicadas. Esta afirmação aplica-se tanto aos aspectos práticos do trabalho, com a necessidade de implementação de ferramentas e aplicação de tecnologias que permitissem obter os resultados pretendidos, como aos seus aspectos mais teóricos, pela amplitude e sobreposição de áreas de estudo que a temática dos Sistemas de Diálogo abarca;
- Sentimos a importância de termos apresentado o nosso trabalho numa conferência de relevo e de o vermos publicado.

Na próxima secção apontamos algumas direcções para possíveis trabalhos futuros, que poderão dar continuidade ao trabalho que apresentámos ao longo desta dissertação.

7.1 Trabalhos futuros

Consideramos nesta secção aspectos relacionados com o próprio Midiki, com a metodologia e com a ferramenta de autoria, sem esquecermos o seu enquadramento em temas mais gerais (ex.: as abordagens à Gestão do Diálogo, os Gestores de Diálogo e os Sistemas de Diálogo em geral). Algumas possibilidades de trabalhos futuros com vista à extensão, aperfeiçoamento e actualização do trabalho realizado, incluem:

- Actualização do Midiki de forma a implementar o *Extended ISU*, com vista ao suporte à interacção multimodal;
- Integração do Midiki num ambiente distribuído via *Open Agent Architecture* (OAA) ou *Galaxy Communicator*, com componentes de reconhecimento de fala robustos e maior facilidade de integração de componentes para outras modalidades;
- Inclusão de um dicionário de sinónimos que permita a um autor seleccionar as palavras que pretende como sinónimos em vez de ter as inserir;
- Disponibilização de suporte nativo à verificação de consistência de respostas do utilizador;
- Disponibilização de funcionalidade que permitam que as respostas ou indicações do utilizador possam ser comparadas com valores obtidos de uma

fonte em tempo real (ex. uma base de dados), em vez de serem comparadas apenas com atributos definidos de forma estática;

- Implementação de um *parsing* mais robusto aos *inputs* do utilizador (em particular no agente *InterpretAgent*). Aperfeiçoar igualmente a geração de *outputs* do sistema (em particular no agente *GenerateAgent*);
- Suporte à totalidade das estratégias possíveis no Midiki, em particular a possibilidade de efectuar algumas parametrizações menos comuns para as estratégias;
- Disponibilização de funcionalidade que permitam abrir e editar um projecto unicamente a partir da sua representação XML. Na versão actual da ferramenta os projectos são editados sempre através do acesso à base de dados. A “exportação” de um diálogo para XML poderia ser realizada de forma relativamente simples recorrendo à mesma tecnologia utilizada para a geração de Java (a JET). A importação a partir de um ficheiro XML pode ser realizada recorrendo a um *parser* SAX (Simple API for XML) [Megginson, 2004] ou a um *parser* DOM (Document Object Model) [W3C, 2005]. Seria também útil poder-se copiar e colar objectos dentro da própria aplicação (ex. planos e estratégias), e;
- Análise e implementação de soluções e/ou ferramentas para a importação e exportação/conversão de informação relativa à modelação de diálogos de/para outros GDs (por exemplo compilações de interacções típicas para serviços com elevado potencial de interesse em sistemas de diálogo, tais como diálogos na área da domótica; diálogos para sistemas de controlo e áudio dentro de veículos/automóveis; etc).

Desta forma concluímos este projecto. Foram tantas as “pequenas” preocupações e sub-objectivos, que nunca deixámos de estar inquietos durante todo este tempo. Acreditamos que foram cumpridos integralmente os objectivos a que nos propusemos.

Publicações do autor

Quintal, L. & Sampaio, P. (2007). A Methodology for Domain Dialogue Engineering with the Midiki Dialogue Manager. In V. Matousek & P. Mautner (Eds.), *Text, Speech and Dialogue, Proceedings of the 10th International Conference TSD 2007*, Lecture Notes in Artificial Intelligence, vol. 4629 (pp. 548–555). Pilsen, Czech Republic: Springer. ISBN: 978-3-540-74627-0.

Referências

Allen, J. F., Byron, D. K., Dzikovska, M., Ferguson, G., Galescu, L. & Stent, A. (2001). Towards Conversational Human-Computer Interaction. *AI Magazine*, 22, pp. 27-37.

Allen, J. F., Ferguson, G. & Stent, A. (2001b). *An architecture for more realistic conversational systems*, Intelligent User Interfaces , Rochester University.

Amores, G., Blaylock, N., Ericsson, S., Pérez, G., Georgila, K., Kaisser, M., Larsson, S., Lemon, O., Manchón, P. & Schehl, J. (2006). *Extended Information State Modeling*. (TALK project Deliverable D3.1). Report: <http://www.talk-project.org/>.

Armstrong, S., Pallotta, V., Clark, A., Coray, G., Georgescu, M., Popescu-Belis, A., Portabella, D., Rajman, M. & Starland, M. (2003). Natural Language Queries on Natural Language Data: a Database of Meeting Dialogues. In A. Dusterhoof & B. Thalheim (Eds.), *Natural Language Processing and Information Systems, 8th International Conference on Applications of Natural Language to Information Systems* (pp. 14-27). Burg (Spreewald), Germany.

Beckham, J. L., Di Fabbrizio, G. & Klarlund, N. (2001,). *Towards SMIL as a Foundation for Multimodal, Multimedia Applications*, In Proceedings of Eurospeech 2001, (pp. 1363-1367) , Aalborg, Denmark.

Berners-Lee, T. (2008). *The Future of the Web*. Retrieved August 1, 2008 from <http://www.technologyreview.com/Infotech/20943/>.

Bevocal (2002). *BeVocal Café*. Retrieved July 8, 2008 from <http://cafe.bevocal.com/>.

Black, A. W., Clark, R., Richmond, K., King, S. & Zen, H. (2008). *The Festival Speech Synthesis System*. Retrieved November 2, 2006 from <http://www.cstr.ed.ac.uk/projects/festival/>.

Bohlin, P., Cooper, R., Engdahl, E. & Larsson, S. (1999). *Accommodating Information States in Dialogue*, Third International Tbilisi Symposium on Language, Logic and Computation , Batumi, Georgia.

Bohlin, P., Cooper, R., Engdahl, E. & Larsson, S. (1999b). Information States and Dialogues Move Engines. *Electronic Transactions on AI*, 4, pp. 53-71.

Bos, J., Klein, E., Lemon, O. & Oka, T. (2003). DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. In *Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue* (pp. 115-124). Sapporo, Japan.

Branco, G., Almeida, L., Gomes, R. & Beires, N. (2005). FASIL: Porque é a falar que Comunicamos melhor!. *Saber & Fazer Telecomunicações, (PT Inovação)*, pp. 37-46.

Buckley, M. & Benz Müller, C. (2005). System Description: A Dialogue Manager Supporting Natural Language Tutorial Dialogue on Proofs. In *Proceedings of the ETAPS Satellite Workshop on User Interfaces for Theorem Provers (UITP 2005)* (pp. 40-67). Edinburgh, Scotland.

Burke, C., Doran, C., Gertner, A., Gregorowicz, A., Harper, L., Korb, J. & Loehr, D. (2003). Dialogue complexity with portability? Research directions for the Information State approach. In *Proceedings of HLT-NAACL 2003 Workshop on Research Directions in Dialogue Processing*. Edmonton, Alberta, Canada.

Burke, C. (2005). Midiki User's Manual, Version 1.0 Beta. *The MITRE Corporation*, URL: <http://midiki.sourceforge.net/>, pp. 1-58.

Burke, C. (2005b). *Midiki: MITRE Dialogue Kit*. Retrieved Junho 3, 2006 from <http://midiki.sourceforge.net/>.

Camps, S. (2003). ZetaDB Zope product. Retrieved Janeiro 1, 2006 from <http://zetadb.sourceforge.net/>.

Carbonell, N. (2005). Multimodal Interfaces – A Generic Design Approach. In C. Stephanidis (Ed.), *Universal Access in Health Telematics, LNCS Series (Hot Topics)* (pp. 209-223). Germany: Springer.

Cavedon, L. (2005). *Dialogue, Dialogue Modeling and Dialogue Systems (Tutorial)*, Australasian Language Technology Workshop, ALTW'2005 University of Sydney.

Cooper, R. & Larsson, S. (1998). Dialogue moves and information states. In H. C. Bunt & E. C. G. Thijsse (Eds.), *Third International Workshop on Computational Semantics (IWCS-3)* (pp. 398-400).

Cooper, R. (2000). *SIRIDUS Project*. Retrieved Abril 5, 2006 from <http://www.ling.gu.se/projekt/siridus/>.

Core, M. (1997). *Dialog Act Markup in Several Layers*. Retrieved Fevereiro 15, 2006 from <http://www.cs.rochester.edu/research/speech/damsl/RevisedManual/node1.html>.

Coutsoumbas, D. (2007). *DSpeech TTS*. Retrieved Julho 15, 2007 from <http://dimio.altervista.org/eng/>.

Danieli, M. & Manca, G. (2005). *Improving speech interaction with mixed initiative dialogue - Loquendo SDS - Spoken Dialog System (White Paper)*. Retrieved Fevereiro 2, 2007 from http://www.loquendo.com/es/whitepapers/Loquendo_SDS.pdf.

Delgado, R. & Araki, M. (2005). *Spoken, Multilingual and Multimodal Dialogue Systems – Development and Assessment*. Chichester: Wiley.

Denecke, M. (2000). An Integrated Development Environment for Spoken Dialogue Systems. In , *Proceedings of the 18th International Conference on Computational Linguistics, COLING 2000*. Centre Universitaire, Luxembourg: Morgan Kaufmann.

Eclipse (2005). *Eclipse 3.2*. Retrieved Novembro 1, 2005 from <http://www.eclipse.org>.

Eclipse (2007). *Eclipse Modeling Framework Project (EMF)*. Retrieved Abril 17, 2007 from <http://www.eclipse.org/modeling/emf/>.

Ferguson, G. & Allen, J.F. (1998). TRIPS: An Integrated Intelligent Problem-Solving Assistant. In *Proceedings of the 15th National Conference on AI (AAAI-98)* (pp. 26-30). Madison, WI.

Fulton, J. (1998). *Introduction to Zope*. Retrieved Setembro 4, 2005 from http://www.plope.com/Books/2_7Edition/IntroducingZope.stx.

Fulton, J. (2005). *Zope Application Server*. Retrieved Setembro 4, 2005 from <http://www.zope.org/>.

Georgila, K. & Lemon, O. (2004,). *Adaptive Multimodal Dialogue Management Based on the Information State Update Approach*, Sophia Antipolis, France.

Ginzburg, J. (1996). Dynamics and the Semantics of Dialogue. *Language, Logic and Computation, CSLI Lecture Notes, Volume 1*.

Goddeau, D., Meng, H., Polifroni, J., Seneff, S. & Busayapongchaiy, S. (1996). A form-based dialogue manager for spoken language applications. In *Proceedings of ICSLP'96* (pp. 701-704). Philadelphia, USA: IEEE.

Guilhoto, P. & Rosa, S. (2001). Reconhecimento de voz. *Departamewnto de Eng. Informática, Faculdade de Ciências e Tecnologia, Universidade de Coimbra*, pp. 1-16.

HO, D. (2005). *Notepad++*. Retrieved Outubro 4, 2005 from <http://notepad-plus.sourceforge.net>.

Hähle, R. & Larsson, S. (2003). *GoDiS.vma - GoDiS Virtual Movie Advisor*. (Project description). Report: Department of Linguistics, University of Gothenburg, Sweden.

IBM (2004). *XHTML+Voice Profile 1.2*. Retrieved Setembro 1, 2005 from <http://www.voicexml.org/specs/multimodal/x+v/12/>.

Infopedia (2007). *Dicionário de Língua Portuguesa*. Retrieved Julho 29, 2007 from [http://www.infopedia.pt/\\$diálogo](http://www.infopedia.pt/$diálogo).

Infopedia, P. E. (2007b). *Oração*. Retrieved Julho 29, 2007 from <http://www.infopedia.pt/>.

Infopedia (2008). *Princípio de cooperação*. Retrieved Agosto 8, 2008 from [http://www.infopedia.pt/\\$princípio-de-cooperacao](http://www.infopedia.pt/$princípio-de-cooperacao).

Johnston, M., Bangalore, S., Vasireddy, G., Stent, A., Ehlen, P., Walker, M., Whittaker, S. & Maloor, P. (2001). MATCH: an architecture for multimodal dialogue systems. In *Proceedings of ACL '02: 40th Annual Meeting on Association for Computational*

Linguistics (pp. 376-338). Philadelphia, Pennsylvania: Association for Computational Linguistics.

Jung, F. (2000). *XML Backgrounder (White Paper)*. Software AG. Retrieved Abril 20, 2007 from http://www1.softwareag.com/Xml/about/e-XML_Backgrounder_WP03E0700.pdf.

Jurafsky, D. & Martin, J.H. (2006). Dialog and Conversational Agents. In *SPEECH and LANGUAGE PROCESSING - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Capítulo 19 (draft): Prentice Hall.

Kellner, A. (2005). Overview of System Architecture. In W. Minker, D. Bühler & L. Dybkjær (Eds.), *Spoken Multimodal Human-Computer Dialogue in Mobile Environments, Proceedings of the ISCA Tutorial and Research Workshop on Multimodal Dialogue in Mobile Environments, Text, Speech and Language Technology* (pp. 117-132). Kloster Irsee, Germany: Springer.

Kordoni, V., Kay, M., Pinkal, M. & Wolska, M. (2007). *Course on Language Technology II*. Retrieved Junho 26, 2008 from <http://www.coli.uni-saarland.de/courses/late2-07/page.php?id=index>.

Kvale, K., Warakagoda, N. D. & Knudsen, J.E. (2003). Speech centric multimodal interfaces for mobile communication systems. *Teletronikk - Temanummer : Spoken Language Technology*, 2, pp. 104-117.

Larson, J. & Candell, E. (2003). *Multimodal Interaction Requirements*. Retrieved Dezembro 2, 2005 from <http://www.w3.org/TR/mmi-reqs/>.

Larson, J. A. (2005). *Standard Languages for Developing Multimodal Applications*. Unpublished Paper.

Larsson, S. & Cooper, R. (2000). An Information State Approach to Natural Interactive Dialogue. In *Proceedings of LREC-2000 workshop on Natural Interactive Dialogue* (pp. 8–12). Athens, Greece.

Larsson, S., Ljunglf, P., Cooper, R., Engdahl, E. & Ericsson, S. (2000b). GoDiS - an accommodating dialogue system. In *Proceedings of ANLP/NAACL-2000 Workshop on Conversational Systems* (pp. 7-10). Seattle, Washington.

Larsson, S., Berman, A., Hallenborg, J. & Hjelm, D. (2004). Trindikit 3.1 Manual. *Department of Linguistics, Gothenburg University*, URL: <http://www.ling.gu.se/projekt/trindi/trindikit/documentation.html>.

Larsson, S. (2005). Dialogue Systems: Simulations or Interfaces?, *In Proceedings of the 9th workshop on the semantics and pragmatics of dialogue, DIALOR'05*. In C. Gardent & B. Gaiffe (Eds.). Nancy, France.

Lemon, O. (2008). Taking computer chat to a whole new level (TALK project). *Research EU - Results Supplement, CORDIS Unit, UE Publications Office*, p. 25.

Ljunglof, P. (2000,). *Formalizing the Dialogue Move Engine*, 4th Workshop on the Semantics and Pragmatics of Dialogue, GÖTALOG 2000, Göteborg, Sweden.

Loquendo, L. (2005). *Loquendo Spoken Dialog System 6.0 (White Paper), Version 1.0*. Retrieved Fevereiro 2, 2007 from <http://www.loquendo.com/en/>.

Martin, D., Cheyer, A. & Moran, D. (1999). The Open Agent Architecture: A framework for building distributed software systems. *Applied Artificial Intelligence: An International Journal*, 13.

McTear, M. F. (1998). Modelling Spoken Dialogues With State Transition Diagrams: Experiences With The CSLU Toolkit. In *Proceedings of the 5th International Conference on Spoken Language Processing, ICSLP-1998* (pp. 1223-1226). Sydney, Australia.

McTear, M. F. (2002). Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Survey*, 34, pp. 90-169.

Megginson, D. (2004). *SAX: Simple API for XML*. Retrieved Março 1, 2007 from <http://www.saxproject.org/>.

Melichar, M. (2005). *Template Driven Dialogue Management Approach in the Framework of Multimodal Interaction*. Unpublished doctoral dissertation, (Ph.D. thesis proposal), École Polytechnique Fédérale de Lausanne (EPFL).

Melichar, M. (2008). *Design of Multimodal Dialogue-Based Systems*. Unpublished doctoral dissertation, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne.

Microsoft (1997-2006). *Microsoft Frontpage*. Retrieved Outubro 4, 2005 from <http://www.microsoft.com/frontpage>.

Microsoft (2007). *Microsoft Speech*. Retrieved Julho 15, 2007 from <http://www.microsoft.com/speech/default.aspx>.

Microsoft (2008). *Microsoft Language Development Center*. Retrieved Julho 4, 2008 from <http://www.microsoft.com/portugal/MLDC/default.aspx>.

Miller, G. (2006). *About WordNet*. Retrieved Setembro 5, 2005 from <http://wordnet.princeton.edu/>.

MITRE, Corporation (2003). *Galaxy Communicator*. Retrieved Abril 24, 2006 from <http://communicator.sourceforge.net/index.shtml>.

Netscape (1995). *JavaScript*. Retrieved August 1, 2008 from <http://en.wikipedia.org/wiki/JavaScript>.

OMG (1997). *UML - Unified Modeling Language*. Retrieved Janeiro 10, 2008 from <http://www.uml.org>.

Ousterhout, J. (1988). *TCL - Tool Command Language*. Retrieved Julho 20, 2008 from <http://en.wikipedia.org/wiki/Tcl>.

Oviatt, S. L. (2002). Breaking the Robustness Barrier: Recent Progress on the Design of Robust Multimodal Systems. *Advances in Computers*, 56, pp. 306-343.

O'Neill, I., Hanna, P., Liu, X., Greer, D. & McTear, M. (2005). Implementing advanced spoken dialogue management in Java. *Science of Computer Programming*, 54, pp. 99-124.

Pallotta, V. & Ballim, A. (2001). Robust Dialogue Understanding in HERALD. In *Proceedings of RANLP 2001 - EuroConference on Recent Advances in Natural Language Processing*. Tzigov-Chark, Bulgaria.

Popma, R. (2004). *JET Tutorial Part 1 (Introduction to JET)*. Retrieved Abril 16, 2007 from http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html.

Powell, A. (2004). *Model with the Eclipse Modeling Framework, Part 1: Create UML models and generate code*. Retrieved Abril 17, 2007 from <http://www.ibm.com/developerworks/opensource/library/os-ecemf1/>.

Quarteroni, S. & Manandhar, S. (2007). A Chatbot-based Interactive Question Answering System. In R. Artstein & L. Vieu (Eds.), *Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue, Decalog 2007* (pp. 83–90). Trento, Italy.

Quesada, J. F., Amores, G., Fernández, G., Bernal, J. A. & López, M.T. (2000). Design constraints and Representation for Dialogue Management in the Automatic Telephone Operator Scenario. In M. Poesio & D. Traum (Eds.), *Proceedings of the 4th Workshop on the Semantics and Pragmatics of Dialogue, GÖTALOG 2000* (pp. 137-142). Göteborg, Sweden.

Quintal, L. & Sampaio, P. (2007). A Methodology for Domain Dialogue Engineering with the Midiki Dialogue Manager. In V. Matousek & P. Mautner (Eds.), *Text, Speech and Dialogue, Proceedings of the 10th International Conference TSD 2007*, Lecture Notes in Artificial Intelligence, vol. 4629 (pp. 548–555). Pilsen, Czech Republic: Springer. ISBN: 978-3-540-74627-0.

Robinson, K., Horowitz, D., Bobadilla, E., Lascelles, M. & Suarez, A. (2004). Conversational Dialogue Management in the FASiL project. In M. Strube & C. Sidner (Eds.), *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue* (pp. 19-22). Cambridge, Massachusetts, USA: ACM Press.

Romellini, C. (2003). *Loquendo Studio: a VoiceXML developer tools suite*. Retrieved Julho 29, 2007 from http://www.voicexmlreview.org/Jan2003/features/Jan2003_loquendo1.html.

Rudnický, A. I. (2005). Multimodal Dialogue Systems. In W. Minker, D. Bühler & L. Dybkjær (Eds.), *Spoken Multimodal Human-Computer Dialogue in Mobile Environments, proceedings of the ISCA Tutorial and Research Workshop on Multimodal Dialogue in Mobile Environments, Text, Speech and Language Technology* (pp. 3-11). Kloster Irsee, Germany: Springer.

Schmidt, G. N. (2005-2008). *XML Copy Editor*. Retrieved Setembro 6, 2006 from <http://xml-copy-editor.sourceforge.net>.

Seneff, S., Lau, R., Pao, C. & Zue, V. (1998). Galaxy-II: A reference architecture for conversational system development. In *Proceedings of the 5th International Conference on Spoken Language Processing, ICSLP 98* (pp. 931-934). Sydney, Australia.

Sparx (2000-2008). *Enterprise Architect*. Retrieved Setembro 14, 2005 from <http://www.sparxsystems.com.au>.

Sperberg-McQueen, C. M. & Thompson, H. (2000). *XML Schema, W3C Architecture Domain*. Retrieved Setembro 2, 2005 from <http://www.w3.org/XML/Schema.html>.

Stein, A. & Thiel, U. (1993). A Conversational Model of Multimodal Interaction. In *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI'93* (pp. 283-288). Washington DC, USA: AAAI Press/The MIT Press.

SUN (1994). *Java*. Retrieved Janeiro 1, 2007 from <http://java.sun.com>.

TecnoVoz (2008). *TECNOVOZ Tecnologia de Reconhecimento e Síntese de Voz*. Retrieved Julho 4, 2008 from <http://www.tecnovoz.pt>.

Traum, D., Bos, J., Cooper, R., Larsson, S., Lewin, I., Matheson, C. & Poesio, M. (1999). *A model of dialogue moves and information state revision*. (TRINDI project deliverable D2.1). Report: University of Gothenburg, Sweden.

Traum, D. & Larsson, S. (2000). Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 6, pp. 323-340.

Traum, D. & Larsson, S. (2003). The Information State Approach to Dialogue Management. In R. Smith & J. Kuppevelt (Eds.), *Current and New Directions in Discourse & Dialogue*. : Kluwer Academic Publishers. pp. 325-353.

Trung, H. B. (2006). *Multimodal Dialogue Management*. (State of the art mini-report). Twente: University of Twente.

Vanderheiden, G., Zimmerman, G., Blaedow, K. & Trewin, S. (2005). Hello, What Do You Do? Natural Language Interaction with Intelligent Environments. In *Proceedings of HCII 2005* (pp.). Las Vegas, USA.

VoiceXML (2004). *VoiceXML SPECIFICATIONS*. Retrieved Setembro 3, 2007 from <http://www.voicexml.org/spec.html>.

W3C (2003). *EMMA: Extensible MultiModal Annotation markup language*. Retrieved Dezembro 2, 2005 from <http://www.w3.org/TR/emma/>.

W3C (2005). *Document Object Model (DOM)*. Retrieved Março 1, 2007 from <http://www.w3.org/DOM/>.

W3Schools (2007). *DTD Tutorial*. Retrieved Fevereiro 17, 2007 from <http://www.w3schools.com/DTD/>.

Wahlster, W., Reithinger, N. & Blocher, A. (2001b). Smartkom: Towards multimodal dialogues with anthropomorphic interface agents. In *Proceedings of MTI Status Conference* (pp. 22-34). Saarbrücken, Germany.

Wang, K. (2002). SALT: a Spoken Language Interface for Web-Based Multimodal Dialog Systems. In *Proceedings of 7th International Conference on Spoken Language Processing, ICSLP-2002* (pp. 2241-2244). Denver, Colorado, USA.

Webyog (2001). *SQLyog*. Retrieved Outubro 3, 2005 from <http://www.webyog.com>.

Wikipedia (2007). Retrieved Fevereiro 7, 2007 from <http://en.wikipedia.org/wiki/Model-view-controller>.

Wikipedia (2006). *Máquina de estado finito*. Retrieved Janeiro 27, 2006 from http://pt.wikipedia.org/wiki/M%C3%A1quina_de_estado_finito.

Wilks, Y., Catizone, R. & Turunen, M. (2006). *Dialogue Management*. (Companions EU 6th-FP IST project). State Of The Art Paper.

Wilson, M. D. (1991). *An Architecture for Multimodal Dialogue*, Venaco 2nd Multi-Modal Workshop, Corsica, France.

van Rossum, G. (1991). *About Python*. Retrieved Setembro 4, 2005 from <http://www.python.org/about/>.

Anexos

ANEXO I – XML Document Type Definitions (DTDs)

i) DTD relativo à componente declarativa *domínio*

```
<!ELEMENT MIDIKIDOMAIN
(name,project,descr?,initializeAttributes,initializeTasks,initializeQu
estions)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT project (#PCDATA)>
<!ELEMENT descr (#PCDATA)>

<!ELEMENT initializeAttributes (attribute+)>
<!ELEMENT attribute (attributeType,descr?,value+)>
<!ELEMENT attributeType (#PCDATA)>
<!ELEMENT value (#PCDATA)>

<!ELEMENT initializeTasks (defaultTask,plan+)>
<!ELEMENT defaultTask (#PCDATA)>
<!ELEMENT plan (name,descr?,baseCell,isSubPlan,strategy+)>
<!ELEMENT baseCell (#PCDATA)>
<!ELEMENT isSubPlan (#PCDATA)>

<!ELEMENT strategy
(name,type,descr?,baseCell?,(move,value?)*,(guard,value,subPlan,subPla
n?)*,position)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT move (#PCDATA)>
<!ELEMENT guard (#PCDATA)>
<!ELEMENT subPlan (#PCDATA)>
<!ELEMENT position (#PCDATA)>

<!ELEMENT initializeQuestions (descr?,defaultQuestion,questionType+)>
<!ELEMENT defaultQuestion (#PCDATA)>
<!ELEMENT questionType (strategyName,attributeType)>
<!ELEMENT strategyName (#PCDATA)>
```

ii) DTD relativo à componente declarativa *léxico*

```
<!ELEMENT MIDIKILEXICON
(name,project,descr?,initializeOutputMatches+,initializeSynonyms*,init
ializeInputMatches*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT project (#PCDATA)>
<!ELEMENT descr (#PCDATA)>

<!ELEMENT initializeOutputMatches (outputMatcher+)>
<!ELEMENT outputMatcher
(inConstruct,position,inMoveType,inStrategy?,outText,outConstruct,outT
ext?,repeatMessage?)>
<!ELEMENT inConstruct (#PCDATA)>
<!ELEMENT position (#PCDATA)>
<!ELEMENT inMoveType (#PCDATA)>
<!ELEMENT inStrategy (#PCDATA)>
<!ELEMENT outText (#PCDATA)>
<!ELEMENT outConstruct (#PCDATA)>
<!ELEMENT repeatMessage (#PCDATA)>

<!ELEMENT initializeSynonyms (addSynonyms+)>
<!ELEMENT addSynonyms (type,word*)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT word (value,synonym+)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT synonym (#PCDATA)>

<!ELEMENT initializeInputMatches (inputMatcher+)>
<!ELEMENT inputMatcher
(inToken,position,outMove?,outStrategy?,outValue?,outVar?,outConstruct
)>
<!ELEMENT inToken (#PCDATA)>
<!ELEMENT outMove (#PCDATA)>
<!ELEMENT outStrategy (#PCDATA)>
<!ELEMENT outValue (#PCDATA)>
<!ELEMENT outVar (#PCDATA)>
```

iii) DTD relativo à componente declarativa *cells*

```
<!ELEMENT MIDIKICELL
(name,project,descr?,addSlots+,addQueries?,addMethods?,initializeHandl
ers?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT project (#PCDATA)>
<!ELEMENT descr (#PCDATA)>

<!ELEMENT addSlots (slot+)>
<!ELEMENT slot (name,type)>
<!ELEMENT type (#PCDATA)>

<!ELEMENT addQueries (query*)>
<!ELEMENT query (strategy,descr?)>
<!ELEMENT strategy (#PCDATA)>

<!ELEMENT addMethods (method*)>
<!ELEMENT method (strategy,descr?)>

<!ELEMENT initializeHandlers (handler*)>
<!ELEMENT handler (strategyName,handlerType,javaClass)>
<!ELEMENT strategyName (#PCDATA)>
<!ELEMENT handlerType (#PCDATA)>
<!ELEMENT javaClass (#PCDATA)>
```

ANEXO II – Os *dialogue moves* e as estratégias no Midiki

Dialogue moves

Um *dialogue move* [Ginzburg, 1996]; [Cooper & Larsson, 1998]; [Traum *et al.*, 1999]; [Bohlin *et al.*, 1999b]; [Larsson *et al.*, 2004]; [Quarteroni & Manandhar, 2007] constitui um acto ou uma acção associado a uma vocalização, no contexto de um diálogo. Pressupõe um conteúdo informativo e uma intenção de comunicação de uma expressão (ou de parte de uma expressão) [Burke, 2005]; [Stein & Thiel, 1993]. Os *dialogue moves* permitem criar uma abstracção entre o grande número e variedade de expressões possíveis, em particular em linguagem natural, e os tipos de actualizações do estado a realizar, tendo em conta as expressões produzidas [Traum *et al.*, 1999].

No Midiki, cada oração (em texto) recebida do utilizador é convertida num ou mais *dialogue moves* pelo *InterpretAgent*. Reciprocamente, cada saída do sistema para o utilizador é primeiramente expressa em termos de *dialogue moves* e só depois é que é expressa em linguagem natural para lhe ser apresentada (esta segunda conversão é da responsabilidade do *GenerateAgent*).

Não obstante os *dialogue moves* constituírem parte integrante da abordagem ISU, o seu tipo e número varia consoante a implementação do gestor de diálogo em si, em particular depende da forma como as regras DME são implementadas.

Por omissão, o Midiki utiliza um conjunto de *moves* baseado nos *moves* do sistema GoDiS [Larsson *et al.*, 2000b]; [Traum & Larsson, 2000]. A Tabela 11 apresenta os *dialogue moves* que podem ser obtidos no Midiki a partir dos *inputs* do utilizador.

Estratégias

Ao contrário do que acontece com os *dialogue moves*, que podem ocorrer tanto em entradas como em saídas do GD, e ser classificados de várias formas, as estratégias ocorrem apenas numa situação: são inseridas pelo autor nos planos criados para o diálogo.

No Midiki, algumas estratégias possuem carácter conversacional, no sentido em que contribuem de forma visível para o progresso do diálogo entre o utilizador e o sistema, introduzindo novo conteúdo no diálogo (ex.: *findout* ou *inform*). Outras estratégias são utilizadas para obter e processar informação de *back end* e para gerir a progressão do plano. Estes elementos não linguísticos incluem iterações, elementos de execução condicional, invocação de subplanos ou invocação de acções de sistema. Podemos chamar-lhes estratégias não conversacionais (ex.: *queryCall* ou *ifThen*).

Planos, Estratégias e *Dialogue moves*

Embora um diálogo seja modelado com planos e estratégias bem definidos, a sua representação dinâmica (ou seja, o plano em execução) não pode ser prevista de forma determinista, uma vez que até que a necessidade de informação de uma estratégia esteja satisfeita pode ser gerado um número indefinido de *dialogue moves*, em particular em estratégias do tipo *findout*. Por exemplo um *findout* associado à pergunta “Qual o destino pretendido?”, pode implicar uma alternância indefinida de perguntas e respostas até que uma das respostas seja aceite e integrada como a resposta àquela estratégia. A execução de uma estratégia pode implicar apenas uma questão (do sistema ao utilizador humano) e uma resposta (do utilizador), ou várias questões e várias respostas, com ou sem repetição das mesmas perguntas, ie, podem ser colocadas questões alternativas (embora com o mesmo objectivo) se as respostas obtidas sucessivamente não forem consideradas válidas pelo sistema. Em resumo, cada estratégia dá origem a um ou vários *moves*, eventualmente em várias alternâncias, até que a questão em discussão tenha sido esclarecida, ou seja, integrada no estado do diálogo.

A Tabela 10 apresenta as estratégias suportadas pela ferramenta implementada e os correspondentes *dialogue moves* desencadeados. As estratégias *Findout*, *Inform*, *Assert* (mais os *moves reqRep* e *Ack*) são do tipo conversacional, ao passo que as estratégias *queryCall*, *methodCall*, *ifThen*, *ifThenElse* e *Exec* são do tipo não conversacional. A “estratégia” *addMove* pode ser considerada genérica, uma vez que tanto pode integrar *moves* conversacionais como *moves* não conversacionais.

Tabela 10: Descrição das estratégias suportadas e respectivos *dialogue moves* de saída

Estratégia	Descrição
<i>Findout</i>	As estratégias <i>findout</i> são utilizadas para obter informação do utilizador. Um <i>findout</i> despoletará sempre um <i>move</i> do tipo <i>ask(questão)</i> (por parte do sistema) para obter normalmente um <i>move</i> do tipo <i>answer(resposta)</i> (por parte do utilizador). Pode acontecer que no decorrer no diálogo o utilizador forneça informação relevante sem que esta lhe tenha sido explicitamente solicitada, pelo que nesse caso o sistema “acomoda” essa informação para utilização posterior

	[Bohlin <i>et al.</i> , 1999]; [Bohlin <i>et al.</i> , 1999b]. Num <i>move</i> do tipo <i>ask(Q)</i> , a questão “Q” é apresentada ao utilizador. O <i>move</i> é “capturado” por um <i>outputMatch</i> correspondente ²⁰ , que lhe atribui conteúdo lexical (um valor para “Q”).
<i>Inform</i>	Desencadeia um <i>move</i> do tipo <i>inform(proposição)</i> . As estratégias <i>inform</i> são utilizadas precisamente para informar o utilizador (ex.: confirmar um valor ou apresentar uma mensagem). Num <i>move</i> do tipo <i>inform(P)</i> , a proposição “P” é apresentada ao utilizador. “P” não consiste necessariamente numa expressão estática, sendo que normalmente, a um texto base, são acrescentados conteúdos gerados/obtidos dinamicamente no decorrer do diálogo.
<i>Assert</i>	Esta estratégia não desencadeia nenhum <i>move</i> visível para o utilizador, mas permite atribuir um valor a um <i>slot</i> (de uma estratégia <i>findout</i>), o que corresponde a uma proposição. Uma variante de <i>assert</i> permite criar um <i>slot</i> sem que lhe seja associada uma estratégia <i>findout</i> . Esse <i>slot</i> pode guardar um valor, o qual pode ser acedido por um <i>handler</i> .
<i>queryCall</i> e <i>methodCall</i>	Estratégias <i>queryCall</i> e <i>methodCall</i> permitem chamar uma <i>query</i> ou um <i>method</i> num determinado ponto de um plano. São tratadas através de <i>handlers</i> (descritos no capítulo 4) que implementam essas <i>queries</i> ou <i>methods</i> . As estratégias <i>query</i> são normalmente utilizadas para consultas a bases de dados externas e as estratégias <i>method</i> são normalmente utilizadas para o acesso a outras funcionalidades (regras de negócio).
<i>ifThen</i> e <i>ifThenElse</i>	Estratégias <i>ifThen</i> e <i>ifThenElse</i> permitem a execução condicional de um (sub)plano ou (sub)tarefa consoante o valor de uma “guarda”. A guarda pode ser o identificador de um <i>findout</i> ou de um <i>slot</i> genérico.
<i>addMove</i>	Através de <i>addMove</i> podemos inserir num plano alguns <i>moves</i> genéricos: Greet : Corresponde a uma mensagem de saudação ou boas vindas, do sistema para o utilizador. Ocorre normalmente no início do diálogo; Thank : Corresponde a uma mensagem de agradecimento, do sistema para o utilizador; Forget : Corresponde a eliminar (ou esquecer) do histórico todas as proposições definidas ao longo do diálogo, tanto as privadas como as partilhadas. No caso

²⁰ Na forma mais básica, um *move* é identificado por uma tupla da forma <tipo_move, nome_estratégia_associada>.

	<p>concreto do Midiki corresponde a apagar o conteúdo de todos os <i>slots</i>;</p> <p>Forget(Q): Corresponde a eliminar (ou esquecer) do histórico a proposição (pode ser uma resposta dada) correspondente à questão (de um <i>findout</i>) “Q”.</p> <p>Corresponde a apagar o conteúdo do <i>slot</i> associado ao identificador “Q”, e;</p> <p>Quit: Corresponde a uma mensagem de despedida, do sistema para o utilizador. Termina o diálogo.</p>
<i>Exec</i>	<p>Permite executar uma determinada tarefa de forma programática, através da indicação do identificador da respectiva <i>cell/contract</i> de suporte. Pode ser utilizada para parar a execução de uma tarefa e iniciar a execução de outra tarefa separada (sem voltar à tarefa inicial durante a mesma sessão).</p>
Dialogue Move	Descrição
<i>answer</i>	<p>Tem a forma <i>answer(Q,A)</i> e corresponde à situação em que o sistema dá uma resposta “A” a uma questão “Q” do utilizador. Pode ser gerado na sequência de uma estratégia <i>respond</i> existente na agenda do sistema, tal como apresentado na Secção 3.5.1.</p>
<i>reqRep</i>	<p><i>reqRep(R)</i>: significa “request repeat” e ocorre quando é despoletada uma mensagem de pedido de esclarecimento por parte do sistema, dirigida ao utilizador. “R” será a razão para o pedido de esclarecimento, correspondendo normalmente a uma de duas possibilidades: “não entendimento” (o sistema não conseguiu interpretar o mais recente <i>move</i> do utilizador) ou “não relevância” (o sistema não conseguiu integrar o mais recente <i>move</i> do utilizador). Este <i>move</i> de saída é gerado automaticamente pelo sistema.</p>
<i>ack</i>	<p>Neste caso, o <i>move ack(P)</i> corresponde a uma proposição “P” do sistema para o utilizador indicando um reconhecimento ou entendimento (ex. “Entendido, continue por favor.”). Este <i>move</i> de saída é gerado automaticamente pelo sistema.</p>

A Tabela 11 apresenta os *dialogue moves* que podem ser gerados a partir obtidos a partir dos *inputs* do utilizador. Os *moves ask*, *answer* (mais *answerTask*), *reqRep*, *greet*, *quit* e *ack* são do tipo conversacional, ao passo que os *moves failed* e *no_move* são do tipo não conversacional.

Tabela 11: Descrição dos *dialogue moves* de entrada (obtidos a partir dos *inputs* do utilizador)

<i>ask</i>	<p>Ocorre quando uma expressão do utilizador é interpretada como uma questão. No Midiki, este move de entrada ocorre na sequência de um move anterior do sistema do tipo <i>ask</i>(questão) à qual o utilizador responde com uma questão sobre quais as alternativas de resposta possíveis.</p> <p>Note-se que na situação geral em que o utilizador faz um pedido ao sistema, eventualmente através de uma questão, o <i>move</i> relevante a gerar consiste num <i>answerTask</i>, descrito a seguir nesta tabela.</p>
<i>answer</i>	<p>Ocorre quando uma expressão do utilizador é interpretada como uma resposta. Corresponde normalmente a uma resposta do utilizador a uma questão colocada anteriormente pelo sistema através de um <i>findout</i>. Pode também corresponder a uma informação antecipada do utilizador que serve de resposta a uma questão ainda por colocar por parte do sistema.</p> <p>Um <i>answer</i> pode tomar várias formas, consoante o move anterior do sistema ou a resposta do utilizador (ou então a ausência de resposta):</p> <ul style="list-style-type: none"> ○ <i>ask</i>(questão) pelo sistema → <i>answer</i>(resposta) do utilizador; ○ <i>ask</i>(tarefa) pelo sistema → <i>answer</i>(tarefa) do utilizador; ○ <i>ask</i>(questão) pelo sistema → <i>answer</i>(inválida) do utilizador; ○ <i>ask</i>(questão) pelo sistema → <i>answer</i>(não_relevante) do utilizador, ou; ○ “qualquer_move” pelo sistema → <i>answer</i>(tarefa) do utilizador. <p>Na ferramenta definimos um pseudo <i>move answerTask</i>, implementado no código Java com a forma “<i>answer(Task)</i>”, onde “<i>Task</i>” é um identificador de uma tarefa (através do identificador da sua <i>Cell/Contract</i>). A tarefa “<i>Task</i>” será iniciada (ie, o plano principal associado a essa tarefa é iniciado) após este “move”. É normalmente utilizado num <i>inputMatch</i> que, no início do diálogo, identifica qual a tarefa a executar.</p>
<i>reqRep</i>	<p><i>reqRep</i> significa “request repeat” e ocorre quando o <i>input</i> do utilizador é interpretado como um pedido de repetição do último <i>output</i> do sistema (uma questão ou uma informação). Nalgumas situações de erro (ex.: um <i>move failed</i>), a repetição pode ser igualmente o comportamento por omissão adoptado pelo sistema.</p>
<i>greet</i>	<p>Ocorre quando o <i>input</i> do utilizador é interpretado como uma expressão de saudação (ex.: “Olá”).</p>
<i>quit</i>	<p>Ocorre quando é detectada uma expressão de despedida do utilizador dirigida ao sistema, numa expressão de <i>input</i> do utilizador (ex. “Adeus”).</p>

<i>ack</i>	Um move de <i>acknowledgement</i> corresponde a uma expressão do utilizador indicando um reconhecimento ou entendimento (ex. "Ok"). Ocorre normalmente na sequência de um <i>inform</i> (proposição) por parte do sistema, mas também pode ocorrer depois de um outro <i>move</i> qualquer.
<i>failed</i>	Ocorre quando o sistema não consegue tratar o <i>input</i> do utilizador (erro de <i>parsing</i>). Desencadeia um pedido de repetição do mais recente <i>move</i> de saída apresentado ao utilizador.
<i>thank</i>	Um move <i>Thank</i> corresponde a uma expressão do utilizador indicando um agradecimento (ex. "Obrigado"). É normalmente mapeado através de um <i>input match</i> .
<i>forget</i>	Um move <i>Forget</i> corresponde a uma expressão do utilizador indicando que pretende que sejam "esquecidas" todas as entradas (ex. "Recomeçar"). É normalmente mapeado através de um <i>input match</i> .
<i>noMove</i>	Ocorre quando o sistema não consegue identificar qualquer elemento válido no input do utilizador (ex. ausência de resposta por <i>timeout</i>). Nesse caso o sistema apresenta, por omissão, as opções disponíveis.

Concluimos que o conjunto de *dialogue moves* necessário para caracterizar cada expressão do diálogo (ou vocalização, num diálogo falado) é limitado, e isso é relevante pois haverá necessariamente um compromisso entre a variedade de *moves* possíveis e a capacidade de um sistema para diferenciá-los.

ANEXO III – Tipos de construtores relacionados com os *output matches* e os *input matches*

Estes construtores (ou *methods*) ocorrem apenas no código Java gerado, sendo transparentes para o utilizador da ferramenta. A sua descrição neste anexo serve o propósito de permitir uma melhor compreensão do código Java relativo ao léxico, gerado pela ferramenta. Estes *methods* são construtores no sentido em que são os métodos Java utilizados para gerar os resultados de cada *output match* ou *input match*.

No caso dos *output matches*, estes construtores permitem diferenciar os diversos tipos de mensagens que podem ser construídos e apresentados para cada tipo de *move* gerado pelo sistema e “capturado” por um determinado *output match* (por ex.: a apresentação de um texto fixo difere da geração de um texto que integre partes fixas e partes dinâmicas e são utilizados construtores distintos para cada caso) e são utilizados construtores diferentes em cada uma dessas situações. A Tabela 12 apresenta os construtores de entrada para os *output matches*. Estes construtores de entrada permitem que um *move* gerado pelo sistema seja correctamente identificado e capturado por um *output matches* adequado.

Tabela 12: Tipos de construtores de entrada para os *output matches*

Tipo de construtor	Descrição
<i>notDefined</i>	- A utilizar na eventualidade de se pretender indicar um construtor indefinido
<i>InputString</i>	- Utilizado quando existem entradas de <i>moves</i> atómicos, com apenas 1 argumento, como por ex.: <code>addInputString("thank");</code> <code>addOutputTokens("Obrigado pela sua visita");</code> É utilizado essencialmente nos <i>moves</i> genéricos <i>greet</i> , <i>thank</i> e <i>quit</i> . O construtor de saída a associar a este construtor será um <i>OutputTokens</i> .
<i>InputNestedPredicate</i>	- Utilizado com estratégias do tipo <i>findout</i> (para as quais o sistema fará disparar um <i>move</i> do tipo <i>ask</i>) e estratégias do tipo <i>inform</i> com mais de 2 argumentos (incluindo o caso especial <i>informInvalid</i>). São sempre entradas compostas, com 3 argumentos. Ex.: <code>addInputNestedPredicate("ask", "pretendeRegresso", new Variable("X"));</code> ou <code>addInputNestedPredicate("inform", "custoViagem", new Variable("Preco"));</code> No caso em que a variável “X” não seja relevante, mas em que a utilização

<i>InputPredicate</i>	<p>deste construtor é a mais apropriada, é automaticamente criada uma variável muda, como no exemplo seguinte:</p> <pre>addInputNestedPredicate("inform", "viagemConfirmada", new Variable("LocalOMTempVar"));</pre> <p>Não existe um construtor de saída único que seja associado a este construtor. Poderá ser um <i>OutputTokens</i> ou uma combinação de <i>OutputString</i> com <i>OutputVariable</i>.</p> <p>- Utilizado ainda em alguns <i>output matches</i> genéricos: ex.:</p> <pre>addInputPredicate("reqRep", "understanding"); ou addInputPredicate("respond", "_");</pre> <p>O construtor de saída a associar a este construtor será um <i>OutputTokens</i>.</p>
-----------------------	--

A Tabela 13 apresenta os construtores de saída para os *output matches*. Estes construtores de saída permitem construir as mensagens (já na forma de texto) a apresentar ao utilizador, para cada *output match*.

Tabela 13: Tipos de construtores de saída dos *output matches*

Tipo de construtor	Descrição
<i>notDefined</i>	- A utilizar na eventualidade de se pretender indicar um construtor indefinido
<i>OutputVariable</i>	<p>- Utilizado quando há uma variável cujo valor deve ser apresentado ao utilizador, integrado em <i>moves</i> do tipo <i>inform</i>, como por exemplo</p> <pre>addInputNestedPredicate("inform", "custoViagem", new Variable("Preco")); work.addOutputString("A sua viagem custa "); addOutputVariable("Preco"); work.addOutputString(" Euros.");</pre> <p>- Também aparece nalguns <i>output matches</i> genéricos, como por exemplo na indicação de repetição de um <i>move</i> anterior. Ex.:</p> <pre>addInputPredicate("repeat", new Variable("Move1")); addOutputVariable("Move1");</pre> <p>Aparece normalmente associado com <i>OutputString</i>.</p>
<i>OutputString</i>	<p>- Aparece normalmente associado com <i>OutputVariable</i>, para juntar <i>strings</i> fixas à mensagem. Um <i>OutputString</i> precede um <i>OutputVariable</i> e, opcionalmente, poderá também sucedê-lo.</p> <p>- Também é utilizado em saídas de texto estático, como no exemplo seguinte:</p> <pre>addOutputString("Reservamos com sucesso a sua viagem.");</pre>
<i>OutputTokens</i>	- Utilizado com estratégias do tipo <i>findout</i> , nesse caso em associação com o

<p>construtor de entrada <i>InputNestedPredicate</i>, como por ex.:</p> <pre>addInputNestedPredicate("ask", "comoViajar", new Variable("X")); addOutputTokens("Como pretende viajar?");</pre> <p>- Utilizado também quando há um <i>informInvalid</i>, como por ex.:</p> <pre>addInputNestedPredicate("inform", "invalid", new Predicate("nomeTransportadora", args2)); addOutputTokens("Desculpe, não reconheci o nome da transportadora.");</pre> <p>- Utilizado igualmente quando existem <i>inputs</i> atómicos, a seguir a <i>InputString</i> [ex.: <i>quit</i>: <code>addInputString("quit"); addOutputTokens("Obrigado e até à próxima.");</code>]</p>

A Tabela 14 apresenta os construtores de saída dos *input matches*. Estes construtores de saída permitem construir alguns *dialogue moves* que serão passados ao sistema como interpretação de *inputs* específicos do utilizador. O construtor de entrada de um *input match* é sempre *InputTokens*, ou seja, um *input match* recebe sempre uma expressão de texto do utilizador.

Tabela 14: Tipos de construtores de saída dos *input matches*

Tipo de construtor	Descrição
<i>notDefined</i>	- A utilizar na eventualidade de se pretender indicar um construtor indefinido
<i>OutputString</i>	- Utilizado nos <i>input matches</i> com saídas simples (com 1 argumento apenas), como por ex.: <code>addInputTokens("bom dia"); addOutputString("greet");</code> . Corresponde a uma saída de um <i>move</i> simples, com apenas um argumento, que é o tipo de <i>move</i> .
<i>OutputNestedPredicate</i>	- É utilizado nos <i>input matches</i> com saídas compostas (com mais do que dois argumentos). Ex. <code>addInputTokens("partir no dia"); addInputVariable("C"); addOutputNestedPredicate("answer", "diaPartida", new Variable("C"));</code> ou por exemplo <code>addInputTokens("isso mesmo"); addOutputNestedPredicate("answer", "simnao", "sim");</code> - Ainda é utilizado nalguns <i>moves</i> genéricos, como por ex. na indicação de quais as opções ou alternativas: <code>work.addInputTokens("opções"); work.addOutputNestedPredicate("ask", "alts", new Variable("AsOpcoes"));</code>
<i>OutputPredicate</i>	- Utilizado nalguns <i>moves</i> genéricos. Ex. <code>addInputTokens("sim"); addOutputPredicate("answer", "sim");</code>

--	--

Tabelas de *matches* genéricos:

As *strings* de texto apresentadas nas Tabelas 15 e 16 referem-se a *output matches* e a *input matches* genéricos, conforme foi descrito nas Secções 4.3.3; 5.3 e 5.5.

As strings de texto apresentadas são definidas na tabela 'translations' e são configuráveis ao nível do projecto, ie, o texto em si pode ser ajustado a cada diálogo. É incluída uma coluna com a expressão de referência em inglês, mas é o texto da coluna "Texto para expressões do sistema" que é inserido no diálogo. O mesmo se aplica aos *input matches* genéricos.

Tabela 15: Texto genérico a incluir nos *output matches*.

Texto de referência em Inglês	Texto em Português (ou outro idioma) para expressões do sistema	Descrição do <i>output match</i> genérico
The options are:	As opções são:	É utilizado para indicar quais as opções ou alternativas de resposta disponíveis, quando solicitado pelo utilizador
and	e	Estas preposição(ões) são utilizadas quando são apresentadas ao utilizador as várias opções de resposta disponíveis
or	ou	
Do you intend	Pretende	Expressão utilizada quando o sistema pretende determinar qual a tarefa a executar
Please rephrase.	Não interpretei o que disse. Por favor reformule.	Expressão apresentada ao utilizador se o sistema não conseguir determinar o seu <i>input</i> por falta de "entendimento", dando lugar a um <i>move</i> do tipo "request repeat": ("reqRep", "understanding")
What do you mean by that?	O que quer dizer com isso?	Esta expressão que ocorre num contexto semelhante ao anterior, mas neste caso devido a insuficiente "relevância". O <i>move</i> gerado é ("reqRep", "relevance")
What can I do for you?	Em que lhe posso ser útil? Diga 'opções' para saber as 'alternativas'.	Expressão de arranque do diálogo por parte do sistema
Welcome to the travel agency.	Bem-vindo à agência de viagens.	Expressão de boas vindas ao utilizador. Corresponde a um <i>move</i> do tipo <i>greet</i> gerado pelo sistema

Thank you very much.	Muito obrigado.	Agradecimento. Corresponde a um <i>move</i> do tipo <i>thank</i> gerado pelo sistema
Thank you for your visit.	Obrigado pela sua visita!	Expressão de despedida. Corresponde a um <i>move</i> do tipo <i>quit</i> gerado pelo sistema
Sorry, I could not find an answer for your request.	Pedimos desculpa, mas não foi possível encontrar uma resposta para o seu pedido. Por favor contacte os nossos serviços.	Texto a apresentar quando o sistema não consegue corresponder a um pedido do utilizador

As *strings* de texto apresentadas na Tabela 16 são utilizadas para gerar *input matches* genéricos.

Tabela 16: Texto genérico a incluir nos *input matches*.

Texto em Inglês	Texto em Português (ou outro idioma) que pode ser reconhecido nos <i>inputs</i> do utilizador	Descrição do <i>input match</i> genérico (todos os termos indicados nas colunas da esquerda poderão possuir sinónimos)
hello	olá (ou ola, isto a palavra equivalente, sem acento)	Um cumprimento feito pelo utilizador, a que corresponderá a um <i>move</i> do tipo <i>greet</i> recebido pelo sistema
options	opções	É um dos dois termos de base reconhecidos (o outro é “alternativas”) pelo sistema para o utilizado perguntar quais as opções ou alternativas de resposta que tem num dado momento. Corresponderá a um <i>move</i> do tipo “ask alts” recebido pelo sistema
ok	ok	Um termo universal para uma confirmação
goodbye	adeus	Um termo universal para uma despedida. Corresponderá a um <i>move</i> do tipo “quit” recebido pelo sistema
pardon	perdão (ou perdao)	Uma forma universal de solicitar uma repetição do que foi dito anteriormente. Corresponderá a um <i>move</i> do tipo “reqRep” recebido pelo sistema
yes	sim	Concordância. Corresponderá a um <i>move</i> da forma (“answer”, “sim”), qualquer que tenha sido a questão colocada pelo sistema ao utilizador

no	não (ou nao)	Discordância. Corresponderá a um <i>move</i> da forma ("answer", "nao"), qualquer que tenha sido a questão colocada pelo sistema ao utilizador
----	--------------	--

ANEXO IV – A Java Emitter Templates (JET)

A tecnologia utilizada para gerar a codificação em Java do diálogo foi a *Java Emitter Templates (JET)* [Popma, 2004], a qual faz parte do *Eclipse Modelling Framework Project (EMF)* [Powell, 2004]; [Eclipse, 2007]. O EMF é uma *framework* que adopta a referência Model Driven Architecture (MDA), da OMG (Object Management Group). O JET é uma das formas de programação por modelos suportadas pelo EMF permitindo, de uma forma relativamente simples, a implementação de programas que produzem código (Java, XML, etc). O pacote implementado para a geração automática de classes Java para o Midiki (pacote `MidikiAuthoringGenerateJava`) foi exportado para uma biblioteca JAR na qual são disponibilizadas as classes geradoras. São sete as classes Java que podem ser geradas pela biblioteca `MidikiAuthoringGenerateJava.jar`, conforme apresentado na Tabela 17.

Tabela 17: Classes a invocar em `MidikiAuthoringGenerateJava.jar` para gerar as classes Java que especificam um diálogo no Midiki

Passo da metodologia	Componentes e respectivas classes Java geradas	Número de classes geradas	Classe geradora a invocar na biblioteca JAR para obter cada classe
Passos 1 a 3	Domínio	1	<code>Domain2Java</code>
Passos 4 e 5	Léxico	1	<code>Lexicon2Java</code>
Passos 6 a 9	<i>Cells</i>	1 ou mais	<code>Cells2Java</code>
Passo 10	Query/Method handler	0 ou mais	<code>HandlersClass</code>
Passo 11	Tasks	1	<code>TasksClass</code>
Passo 12	DomainAgent	1	<code>DomainAgentClass</code>
Passo 13	“domain executive”	1	<code>dmExecutiveClass</code>

Por exemplo, para obter o código Java que implementa o domínio pode ser invocada a classe `Domain2Java` da seguinte forma, numa linha de comandos que disponibilize e permita a execução o comando “java”: “java -classpath mysql-connector-java-5.0.5-bin.jar;MidikiAuthoringGenerateJava.jar midiki.authoring.Domain2Java > Viagens.java”²¹ para um domínio criado na ferramenta com o nome “Viagens”. O pacote `MidikiAuthoringGenerateJava.jar` reside no servidor e o seu *output* é devolvido ao browser.

²¹ O separador do comando “java” em Windows é “;” e em Linux é “.”

No workspace do Eclipse, este pacote possui a estrutura ilustrada pela Figura 60.

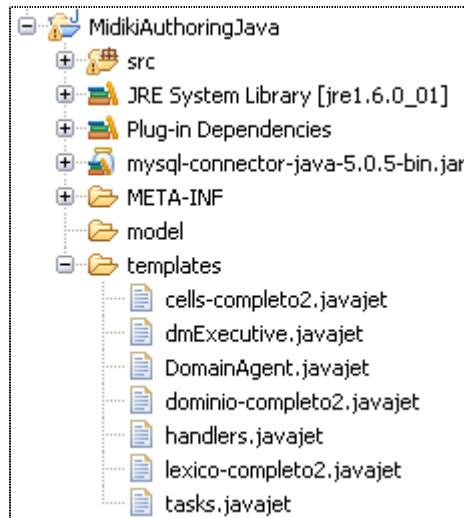


Figura 60: Estrutura de “classes” JET para o pacote MidikiAuthoringGenerateJava

ANEXO V – Modelo Relacional da BD da ferramenta

Por questões de espaço dividimos o diagrama em duas partes: i) parte esquerda e ii) parte direita.

Pretende-se apenas ilustrar o diagrama entidade-relação da base de dados criada e utilizada pela ferramenta de autoria (nem todo o texto da figura é legível).



ii) Lado direito
do diagrama
entidade-
relação

